Yaniel de Sousa Barbosa

Licenciado em Ciências da Engenharia Electrotécnica e Computadores

Modelação e Controlo de Aerogerador em Ambiente de Simulação

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e Computadores

Orientador: Doutor Luís Filipe Figueira de Brito Palma, Professor (Auxiliar), Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Co-orientador: Doutor João Francisco Alves Martins, Professor (Auxiliar), Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri

Presidente: Doutor Luís Augusto Bica Gomes de Oliveira, Professor Auxiliar, FCT-UNL
Arguente: Doutor João Miguel Murta Pina, Professor Auxiliar, FCT-UNL
Vogal: Doutor Luís Filipe Figueira de Brito Palma, Professor Auxiliar, FCT-UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2017

Modelação e Controlo de Aerogerador em Ambiente de Simulação

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Agradecimentos

A realização desta dissertação representa o final e o inicio de uma grande etapa da minha vida, como tal gostaria de agradecer a todos os que estiveram presentes e proporcionaram esta magnifica experiência.

Em primeiro lugar, gostaria de agradecer ao meu Orientador Professor Luís Filipe Figueira Brito Palma e ao meu Coorientador Professor João Francisco Alves Martins pela oportunidade, pela paciência, pelo empenho, pelo conhecimento transmitido, pela atenção durante este último ano e pelos esforços realizados para que houvesse sempre condições para realizar esta dissertação.

Gostaria também de agradecer aos meus colegas Vasco Brito, Jorge Folgado, Alexandre Brito, Liliana Sequeira, Luís Alves e Guilherme Almeida pela disponibilidade, conselhos, palavras de incentivo, assim como todo o auxilio na fase prática deste trabalho.

Um especial agradecimento aos meus pais, José Gomes Barbosa e Ester de Sousa Gustavo, pelas oportunidades proporcionadas ao longo da minha vida e, especialmente nestes últimos 6 anos em que, para além de pais, foram amigos excecionais, conselheiros e tutores extraordinários. Queria também agradecer à minha irmã Yannette Barbosa pelas palavras de motivação e pela sua ajuda.

Aos meus amigos Tiago Romeiro, Francisco Soares, Joel Sousa, João Claro, Rui Ramalho, Daylson Vera Cruz, Idelson Gonzaga, João Guerreiro, Gonçalo Simões, Sónia Pinheiro, Bruno Semedo e Isa Raposo por todas as horas de discussão sobre o tema desta dissertação, pelas ideias, pela motivação, pelo apoio e pelas horas de convívio que contribuíram para elevar o animo aquando a realização deste trabalho.

Um especial agradecimento à minha namorada, companheira e amiga Ana Luísa Martins Maia dos Santos pela paciência, pelo apoio, pela disponibilidade extraordinária ao longo deste último ano.

Gostaria também de agradecer aos Professores Fernando Coito, João Murta Pina, Pedro Pereira e João Rosas, pela disponibilidade e esclarecimento de dúvidas pontuais que

surgiram aquando a implementação. Por fim e não menos importante, gostaria de agradecer aos serviços técnicos pelo seu excelente trabalho e parceria e, à Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa pela oportunidade de aprender, crescer e desenvolver competências que me serão extremamente úteis no meu futuro, tanto a nível profissional como pessoal.

Resumo

A energia eólica é uma fonte de energia renovável e inesgotável, porém possui grandes problemas associados à velocidade e direção do vento. No caso do aerogerador de pequena potência situado no Departamento de Engenharia Electrotécnica – Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, este possui o seu próprio sistema de seguimento do vento, sendo portanto um *free-yaw drive*. Por se mover de acordo com as variações da direção do vento torna-se interessante realizar um estudo sobre a sua produção elétrica real comparativamente à produção elétrica ideal, assumindo que o aerogerador está sempre alinhado com o vento, de forma a perceber quanta energia é perdida nestes aerogeradores de pequena potência, mesmo quando estes possuem um sistema que lhes permite realizar um seguimento mais rápido da direção do vento.

Para realizar este estudo, foi necessário criar sistemas de aquisição de dados de forma a obter amostras da movimentação do aerogerador segundo o ângulo de *Yaw*, conciliar estes dados com a velocidade e direção do vento medida pela estação meteorológica e adquirir dados da potência elétrica. Estes três sistemas foram criados com base em Arduínos Uno, em sensor de medição do ângulo de *Yaw*, em módulos de comunicação RF de forma a manter o sincronismo dos dados e, num Analisador de qualidade de energia de modo a obter a potência elétrica produzida.

Com base nas amostras obtidas em diversos dias de aquisição, foram criados os modelos de potência elétrica ideal e real e, comparados os seus resultados, tendo-se verificado uma perda significativa de energia. De forma a reduzir as perdas, foi criado um novo modelo que permitiu estimar a posição do aerogerador. Foi implementado em ambiente de simulação um sistema de controlo semi-ativo dos travões magnetoreológicos com o intuito de travar e de proporcionar um melhor seguimento da direção do vento, por parte do aerogerador. A posição final do aerogerador foi estimada pelo seu modelo de posição.

Como resultado observou-se que, mesmo em sistemas free-yaw drive, o seguimento da direção do vento e a produção podem ser melhorados com a aplicação de técnicas de controlo semi-ativo, mais especificamente o controlo dos travões magnetoreológicos .

Palavras-chave: Aerogerador, *Yaw*, Vento, Controlo semi-ativo.

Abstract

Nowadays, wind power is being widely used as a renewable source of energy. The major disadvantage is the dependence of the wind's velocity and direction to maximize the power production. This dissertation considers a low-power Horizontal-Axis Wind Turbine, HAWT, installed at the Electrical Engineering Department (Faculty of the Sciences Technology, NOVA University of Lisbon), with the main objective of analysing the differences between the real power production and the ideal power production (assuming that the nacelle is always aligned with the wind direction), based on two low-power HAWT models.

Three acquiring data system were implemented to achieve this goal, obtaining the direction of the nacelle in the yaw angle, the wind velocity and direction from the weather station as well as its produced electrical power. Those systems were created with two Arduíno micro-controllers, a sensor which allows the measurement of yaw movements, RF modules for the synchronization between the different acquiring data systems and a Single Phase Power Quality Analyser to acquire the electrical power produced data.

The ideal and real power model were created based in the acquired data and their production was compared in order to understand how much power was being lost. In a simulation environment, a semi-active magnetic-rheological brake control system was implemented to reduce the yaw error between the nacelle and the wind direction, in order to reduce the losses of the power produced. The yaw error was calculated by a new model, created with the objective to estimate the nacelle position.

It was concluded that, even being a free-yaw drive system, the nacelle could improve its follow-up of the wind direction with the application of the control techniques

Keywords: Wind Turbine, Yaw, Wind, Semi-active control.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas

AM	*Amplitude Modulation*
DC	*Direct current*
DTR	*Data Terminal Ready*
ER	Electroreológico
HAWT	*Horizontal-Axis Wind Turbine*
IMU	*Inertial Measurement Unit*
LTI	*Linear time-invariant*
MIMO	*Multiple-Input-Multiple-Output*
MR	Magnetoreológico
MRF	Fluído Magnetoreológico
PID	*Proportional–Integral–Derivative Controller*
RF	Rádio Frequência
RTS	*Request To Send*
RX	Recetor
SCL	*Serial Clock*
SDA	*Serial Data*

SISO *Sigle-Input-Single-Output*

TX Transmissor

UART *Universal Asynchronous Receiver/Transmitter*

VAWT *Vertical-Axis Wind Turbine*

Lista de Símbolos

AnalogValue	Valor recebido na porta lógica do Arduíno Uno
Area	Área que depende do raio das pás
B	Campo magnético induzido
C(i)	Coeficiente de amortecimento, que depende de I
C_p	Coeficiente de Potência
$\dot{x}$	Variável de estados
η	Rendimento elétrico
F	Somatório das forças
$f_c(H)$	Força de atrito de Coulomb, que varia com H
g	Raio disponível para o fluído
γ	Razão de deformação
$Gyro_X$	Posição angular no eixo do X
$Gyro_Y$	Posição angular no eixo do Y
$Gyro_Z$	Posição angular no eixo do Z
h	Altura a que se deseja estimar a velocidade do vento (Lei Exponencial do cisalhamento do vento)

H	Campo Magnético
H_{TP}	Altitude de medição da temperatura e da pressão do ar
I	Corrente elétrica
J	Momento de Inércia
K_0	Rigidez elástica
K_i	Coeficiente de regressão
m	Massa do objeto em rotação
N_{pr}	Velocidade de rotação do motor em RPM
N_{Pulsos}	Número de pulsos contados
ν	Viscosidade
ω	Velocidade angular
$Pos_{Encoder}$	Posição angular do eixo do Encoder
P_{Pulsos}	Número de pulsos recebidos do Anemómetro
P_{vento}	Potência do vento
R_{espira}	Raio da espira
ρ	Densidade da massa do ar
R_i	Raio interior do disco
R_o	Raio exterior do disco
τ	Tensão de corte

τ_y	Tensão de escoamento
Temp	Temperatura a altura h
T_{fric}	Binário de fricção
θ	Descolamento angular
T_{MR}	Binário Magneto-Reológico
T_{vel}	Período de amostragem para o cálculo da velocidade do vento
v	Velocidade linear do vento
φ	Ângulo entre a direção da turbina e a direção do vento
V_{mph}	Velocidade linear do vento em Miles per hour
x	Deslocamento
z	Variável que descreve a histerese
Z	Altura a que se deseja estimar a velocidade do vento (Lei Logarítmica do cisalhamento do vento)

Capítulo 1

Introdução

1.1 Motivação

A energia eólica tem sido cada vez mais utilizada por ser uma fonte renovável de energia. Apesar de ser uma forma de energia inesgotável, surgem vários problemas associados a esta tecnologia. Fatores como a mudança da velocidade do vento, da direção do vento e as suas influências na energia produzida, são grandes problemas que exigem a necessidade de aplicação de controlo. Este controlo é tipicamente, o controlo do ângulo de Pitch e o alinhamento da turbina com a direção do vento (segundo o ângulo de Yaw). Não sendo o ângulo de Yaw controlado, o aerogerador pode não estar alinhado com a direção do vento, o que pode provocar uma diminuição na energia que este poderia estar efetivamente a produzir. Atualmente, no sistema de seguimento da direção do vento é utilizado um motor de Yaw (*Yaw Drive*), que move o aerogerador de acordo com os dados recebidos da estação meteorológica, ou *Free-Yaw*, em que no caso da HAWT(*Horizontal-Axis Wind Turbine*), situada no Departamento de Engenharia Electrotécnica da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, modelo em que incide o estudo, este seguimento é livre, provocando possivelmente alguma perda de energia produzida, visto que o aerogerador não possui um sistema de travagem na posição de maximização da energia (direção do vento) fornecida pela estação meteorológica.

Refletindo sobre a importância de maximizar essa energia, pretende-se criar dois modelos, um para a produção elétrica real e outro para estimar a produção elétrica ideal do aerogerador. Pretende-se, ainda, comparar a energia produzida pelo modelo real (com base na posição do aerogerador) com a produzida pelo modelo ideal(com base na direção do vento). De forma a corrigir a diferença entre as duas direções (Erro de *Yaw*), pretende-se controlar o ângulo de *Yaw* do aerogerador utilizando um sistema de controlo semi-ativo. Este controlo será realizado com base num travão semi-ativo, para a travagem do

aerogerador no valor de referência que represente a direção do vento.

1.2 Objetivos

Esta Dissertação tem como objetivos a modelação em caixa cinzenta do aerogerador de pequena potência e a comparação entre a produção de energia elétrica produzida, no caso do aerogerador estar alinhado com a direção do vento (Utilização do modelo Ideal do Aerogerador) e no caso real (Utilização do modelo real do aerogerador), Figura 1.1. Tem também como objetivo o desenvolvimento de um sistema de controlo que permita ao aerogerador realizar o seguimento da direção do vento, de forma a maximizar a potência elétrica produzida e, a realização de uma plataforma de monitorização.

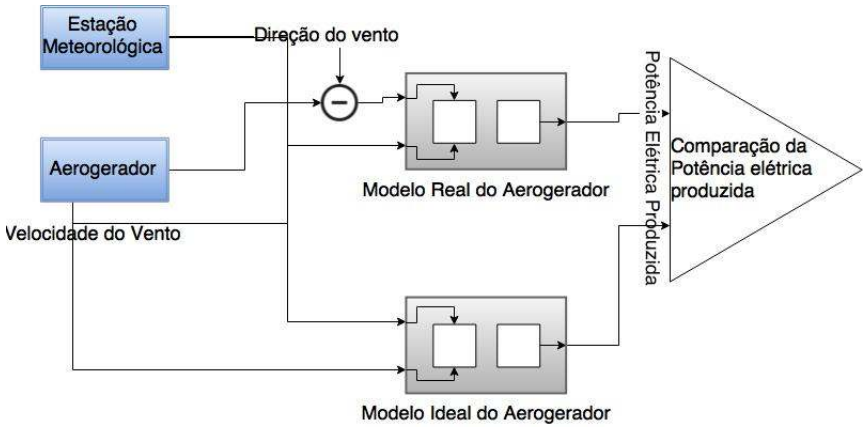


Figura 1.1: Arquitetura de comparação entre a energia produzida pelo modelo real e o modelo ideal do aerogerador.

Após a aplicação do sistema de controlo, pretende-se realizar simulações e comparar as produções com controlo e sem controlo (Figura 1.2).

Os objetivos intermédios associados ao desenvolvimento desta dissertação são:

- validação dos dados dos sensores que serão utilizados nos diferentes Arduínos;
- implementação do sistema de sincronismo entre os sistemas de aquisição de dados;
- aquisição e armazenamento de dados do Aerogerador, da estação meteorológica e da potência elétrica produzida;
- realização da plataforma de monitorização dos dados adquiridos;
- modelação do Aerogerador com base nos dados da sua produção, velocidade e direção do vento obtidos e, os parâmetros conhecidos;
- análise e comparação dos modelos de potência elétrica;
- estudo de diferentes modelos de travões semi-ativos;

- implementação do controlo binário de travagem e do controlo de posição do aerogerador;

- análise, simulação e comparação da potência elétrica produzida com o controlo aplicado.

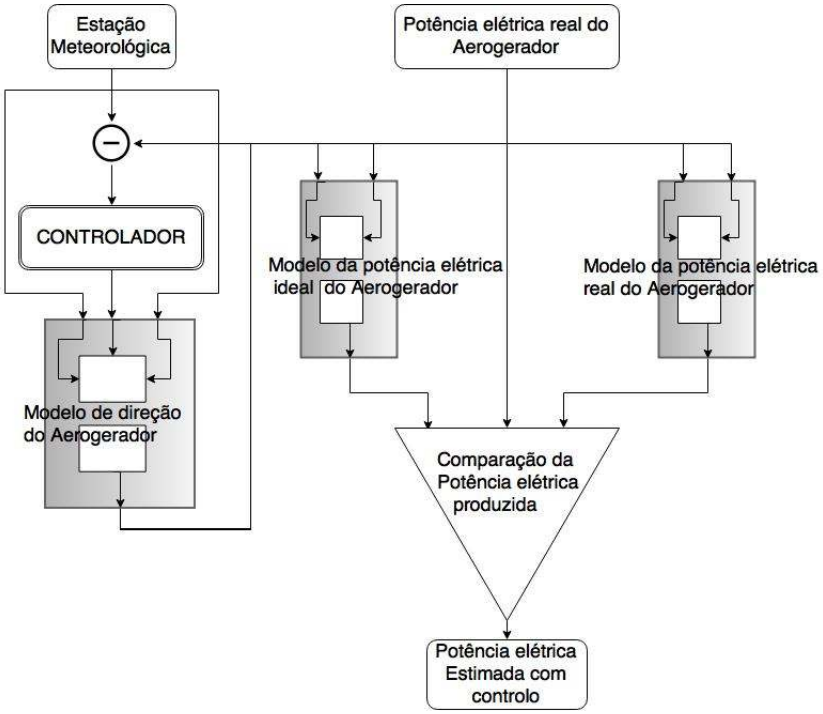


Figura 1.2: Arquitetura de controlo e comparação das produções com controlo e sem controlo.

1.3 Contribuições

Esta dissertação tenciona contribuir com a criação dos modelos reais e ideais da produção elétrica do aerogerador, com a criação e controlo do modelo de orientação e, com os resultados associados ao aumento da potência elétrica produzida. Contribuir também com os sistemas de aquisição de dados da orientação do aerogerador de pequena potência, da estação meteorológica, da potência elétrica produzida, com a plataforma de monitorização, que permita a aquisição, armazenamento e análise dos dados com um tempo de amostragem de aproximadamente 4 segundos (inferior aos aproximadamente 60 do sistema existente) e com um artigo científico intitulado *Modeling and Semi-active control for a low power and free-yaw drive HAWT*, apresentado na conferência YEF-ECE 2017 e publicado pela *IEEE Explore* com o DOI:10.1109/YEF-ECE.2017.7935643.

1.4 Estrutura da Tese

A presente Dissertação encontra-se estruturada em cinco capítulos:

- Capítulo 1, capítulo introdutório onde é explicada a escolha deste tema, o seu propósito e a motivação para a sua implementação.

- O Capítulo 2 inicia-se com uma introdução histórica, de forma a demonstrar o interesse pelo estudo dos sistemas de produção de energia eólica, ao longo dos tempos. Posteriormente são explicados os conceitos que serão utilizados na dissertação e, são referidos alguns trabalhos relevantes para a realização deste trabalho, indicando que tipo de abordagens têm sido implementadas.

- No Capítulo 3 é explicada a teoria associada ao Capítulo 5. São explicados os conceitos e os passos seguidos na implementação do sistema de medição do ângulo de *Yaw* do aerogerador; da aquisição de dados da estação meteorológica; da medição da potência elétrica produzida; da criação do modelo de potência elétrica produzida com base na posição do aerogerador, na direção e na velocidade do vento (Aerogerador-Modelo Real); da criação do modelo de potência elétrica produzida com base na direção e velocidade do vento (Aerogerador- Modelo Ideal). São também referenciados alguns modelos matemáticos que representam o travão Magneto-Reológico, explicando o porquê da sua implementação.

- No Capítulo 5 são implementados os três sistemas de aquisição de dados, o sincronismo entre os sistemas e a unidade central, o sistema de armazenamento de dados, a plataforma de monitorização de dados, os modelos de potência elétrica real e ideal, o modelo da direção do aerogerador e o travão Magneto-Reológico. São também implementados o controlador de binário, o controlador do ângulo de *Yaw* do aerogerador e, é realizada uma simulação da potência elétrica produzida com e sem controlo aplicado. No final deste capítulo efetuada a análise e discussão dos resultados.

- No Capítulo 5 é elaborada a conclusão e são feitas sugestões para possíveis trabalhos futuros.

Capítulo 2

Conceitos

2.1 Introdução Histórica

A produção de energia eólica consiste num método de aproveitamento da energia do vento para a produção de potência, quer seja mecânica ou elétrica. A conversão da energia do vento em energia mecânica iniciou-se à milhares de anos, começando pelos moinhos de vento de eixo vertical, encontrados no limite Persa-Afegão e posteriormente na Europa, com os moinhos de vento de eixo horizontal, desde o século XIV até meados do século XIX (1300-1875). O passo seguinte foi também no século XIX, quando passaram a ser utilizados para bombear água, (Kaldellis e Zafirakis, 2011). O conceito conhecido atualmente, surgiu no final do século XIX (1887-1888) com o crescimento do uso da energia elétrica, quando Charles F.Brush utilizou a primeira turbina eólica em madeira, Figura 2.2, para gerar energia elétrica, (Rüncos et al., 2005).

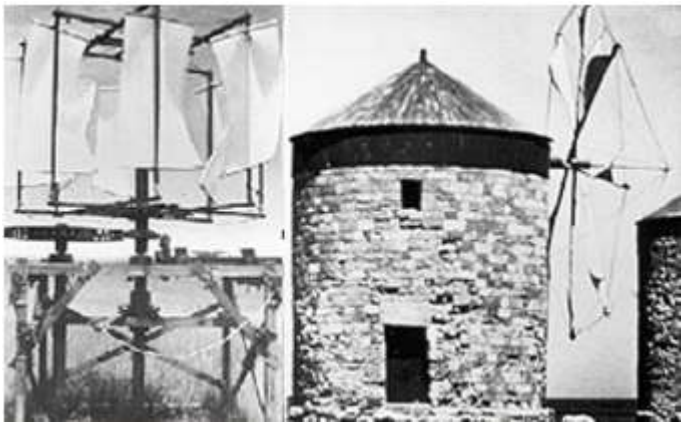


Figura 2.1: Moinho de eixo vertical e horizontal, (Kaldellis e Zafirakis, 2011).

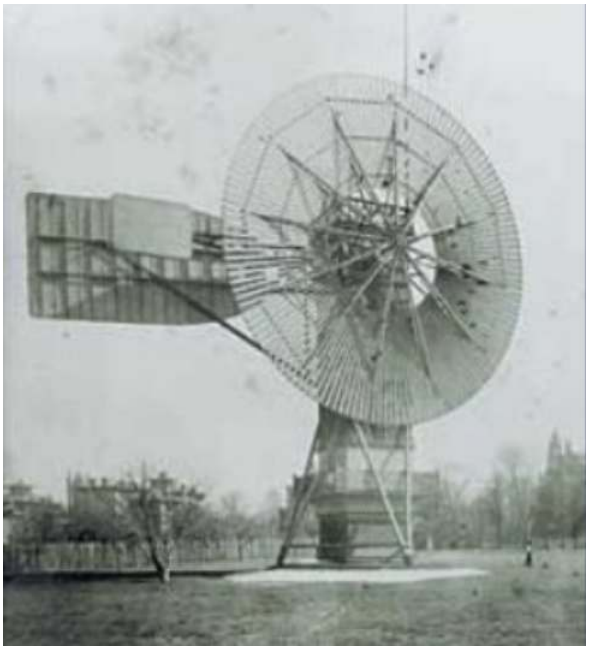


Figura 2.2: Moinho de eixo horizontal em madeira, (Rüncos et al., 2005).

Já na década de 50 do século XX, na Alemanha, o mesmo conceito passou a ser implementado com materiais compostos, métodos de controlo do ângulo de *Pitch* e torres na forma de tubos, denotando-se assim uma evolução.

Comparativamente com a energia não renovável, especificamente com o petróleo, a energia eólica tornou-se, por motivos financeiros, foco de grandes estudos, sendo considerada uma grande contribuição para o sistema de energia elétrica, tanto na produção como na eficiência, gerando competitividade.

Ao longo de todo o desenvolvimento tecnológico conseguiram-se reduções de custos da energia instalada, devido ao aumento da produção e à facilidade de implementação deste sistema, verificando-se essa tendência com o crescimento acentuado da energia produzida nos últimos anos, demonstrado na Figura 2.3 .

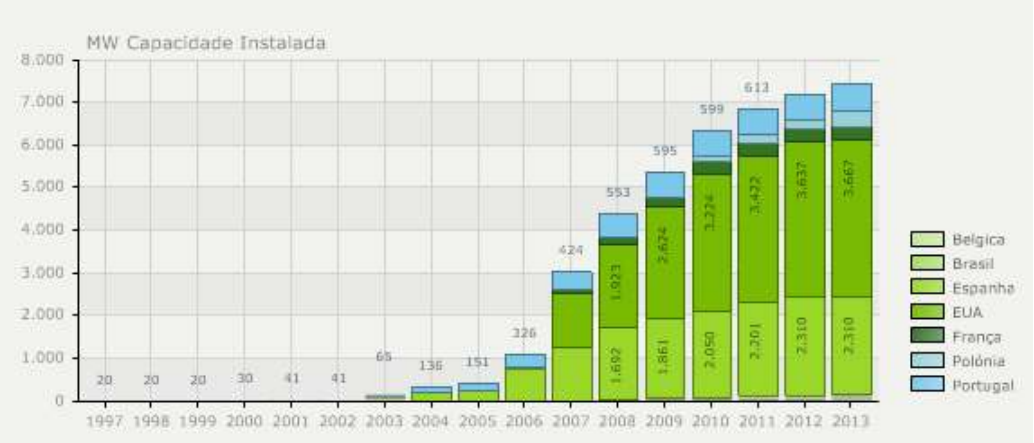


Figura 2.3: Produção anual de energia Eólica, 3/7/2016, (EDP, 2016).

Como se verifica na Figura 2.3 existe um crescimento anual significativo em torno da energia eólica, sendo que, de 2006 a 2013, o aumento da produção ultrapassou os 7MW(Megawatt) anuais. Estes factos demonstram a importância deste tipo de energia, justificando o seu constante estudo e desenvolvimento.

Apesar de limpa, existem fatores que tornam complexa a sua implementação quando comparada com outras fontes de energias renováveis, como por exemplo, a Energia solar (CM-Moura, 2016) e a mini-hídrica, que obtiveram maior crescimento nos últimos anos.

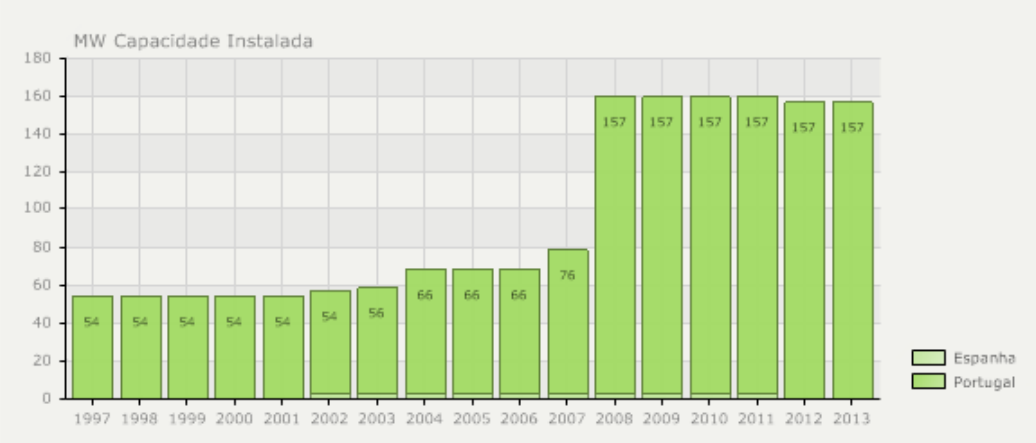


Figura 2.4: Produção anual da Mini-Hídrica, 3/7/2016, (EDP, 2016).

2.2 Funcionamento de um Aerogerador

Figura 2.5: VAWT Darius, (Ferreira, 2011).

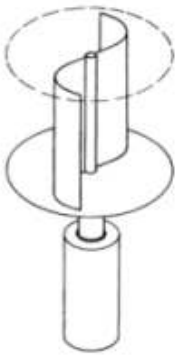


Figura 2.6: VAWT Savonious, (Ferreira, 2011).

Existem dois tipos de turbinas eólicas sendo que a sua designação varia com o eixo de rotação das pás. As conhecidas como VAWT, Figura 2.5 e 2.6, têm a vantagem de não dependerem da direção do vento para extrair energia. Estas têm tipicamente dois designs:

- as turbinas *Darrieus* , que têm como vantagem a sua localização ao nível do solo e a sua utilização em locais com ventos fortes;

- as turbinas *Savonius*, que são usadas quando se pretende gerar pouca energia a um custo e manutenção reduzidas.

Quanto às HAWT, estas dependem da direção e intensidade do vento, da área relativamente ao tamanho das pás, do seu coeficiente de potência e da densidade da massa do ar. São compostas por diversos componentes, Figura 2.7, responsáveis pela conversão de energia eólica em energia elétrica.

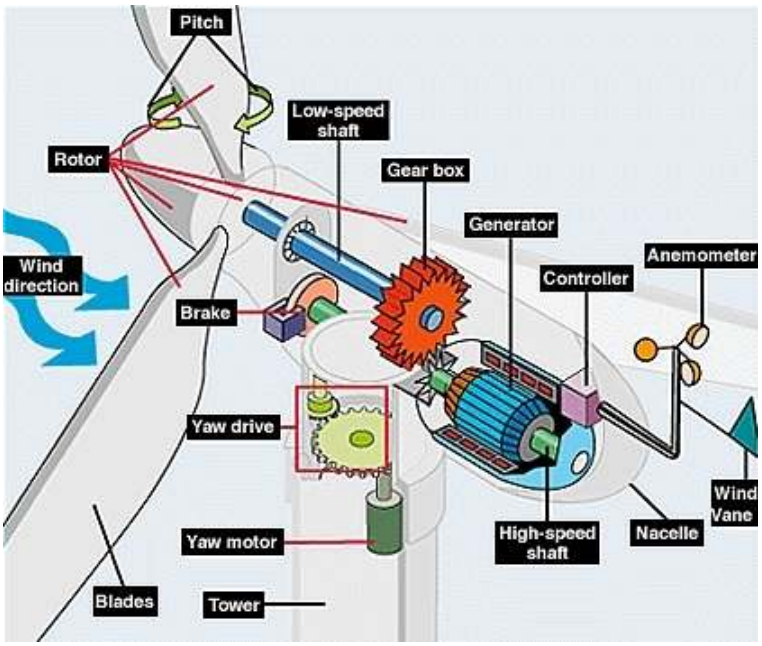


Figura 2.7: Componentes básicas de uma HAWT.

Na Figura 2.7 estão representados os componentes básicos que integram o sistema de produção de energia eólica. A torre que sustenta o aerogerador possui, tipicamente, mais de 80 metros de altura (Stol, 2002), uma vez que a velocidade do vento aumenta quanto maior for a altura, segundo duas leis :

Lei Exponencial do cisalhamento do vento :

$$\frac{v}{v_0} = (\frac{h}{h_0})^{\alpha} \tag{2.1}$$

Sendo α o coeficiente de fricção, h a altura a qual se deseja saber a velocidade, h_0 a altura do Anemómetro, v a velocidade que se deseja saber e v_0 a velocidade medida pelo Anemómetro, (Jonkman, 2009).
Lei Logarítmica do cisalhamento do vento:

$$\frac{v}{v_0} = \frac{1}{k} ln(\frac{Z}{Z_0}) \tag{2.2}$$

Sendo k a constante de Von Karman (aproximadamente 0.4), Z a altura a qual se deseja saber a velocidade, Z_0 o comprimento de rugosidade aerodinâmica, v a velocidade que se deseja saber e v_0 a velocidade medida pelo Anemómetro, (Jonkman, 2009; Roballo et al., 2009).

O gerador é o responsável pela conversão da potência mecânica do eixo de baixa velocidade (*Low-Speed Shaft*) em potência elétrica. O eixo de baixa velocidade pode conectar a *Nacelle* diretamente ao gerador ou, pode ter um multiplicador de velocidade que aumente a velocidade de rotação do eixo à entrada do gerador. O *Yaw Drive* situa-se no topo da torre e permite que o aerogerador siga a direção do vento . O rotor é responsável por capturar a energia cinética do vento, transformando-a em energia mecânica, Equação 2.3, sendo o rendimento da conversão C_p máximo descrito pelo princípio de *Albert Betz*.

$$P_{mec} = \frac{1}{2} C_p A \rho (v \cos(\varphi))^3 \tag{2.3}$$

onde C_p é o coeficiente de potência, $Area$ a área relativamente ao raio das pás, ρ a densidade da massa de ar, v a velocidade do vento e φ o erro de alinhamento da turbina relativamente a direção do vento (*Yaw Error*).

Essa energia mecânica é gerada quando uma densidade da massa de ar atravessa a área da turbina aplicando uma força nas pás, força essa que introduz duas componentes essenciais para a sua rotação, Figura 2.8, (Reddy et al., 2015):

- Componente de sustentação(*Lift*) , perpendicular à direção do vento.
- Componente de arrastamento (*Drag*), paralela ao vento.

Essas componentes são as responsáveis pela rotação das pás gerando uma potência mecânica, que é apenas uma parte da potência real do vento. Essa quantidade de potência conseguida a partir do vento é no fundo um rendimento, que na Equação 2.3, é representado por C_p, e que segundo *Albert Betz* o valor máximo teórico do C_p é $C_{p_{max}} = 59.3\%$.

Como observado na Figura 2.8, para além da intensidade do vento, a rotação depende do ângulo de ataque do vento nas pás, que por sua vez depende do ângulo de Pitch da pá. Este ângulo de Pitch deve ser controlado de forma a otimizar o ângulo de ataque, visto que, caso este aumente, pode dar origem à perda de sustentação, provocando uma travagem nas pás.

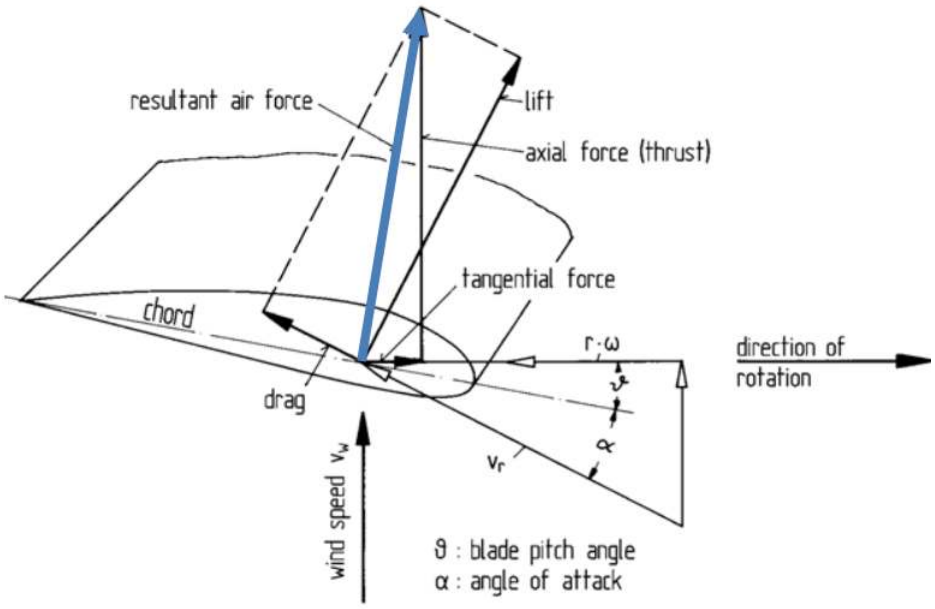


Figura 2.8: Diagrama de forças que atuam na secção de uma pá.

Qualquer comportamento de um sistema de produção de energia eólica é dependente do vento, como tal, surge a necessidade de controlar esse comportamento de forma a se tirar o melhor proveito deste. Atualmente, esse controlo é aplicado essencialmente ao ângulo de *Pitch* e ao ângulo de *Yaw* (Figura 2.9):

- Segundo (Castro, 2005), o controlo do ângulo de Pitch é o que mais influência a produção de energia, pois permite determinar o ângulo de ataque necessário para maximizar a força de atuação do vento nas pás, embora, inicialmente, fosse utilizado apenas para limitar a velocidade de rotação no seu valor máximo. Este permite, também, diminuir a carga experimentada pela turbina, aumentando assim, o seu tempo de vida e mantendo um determinado rendimento.

- O controlo do ângulo de *Yaw* é um tema igualmente importante. Segundo (Ferreira, 2011), para que haja o efeito de turbina e a maximização da conversão do vento em potência mecânica, é necessário que a turbina esteja na posição de *Upwind*, (Scholbrock et al., 2015), ou seja, que o vento venha de frente para a turbina, de forma perpendicular ao plano das pás e em paralelo à turbina. (Kragh e Fleming, 2012) referem que, pelo facto do erro de alinhamento ser um fator que está elevado ao cubo, este proporciona uma redução significativa da potência do vento, à medida que o desalinhamento do rotor em relação ao vento aumenta.

Figura 2.9: Representação do ângulo de Pitch e de Yaw, adaptação do modelo de uma turbina eólica, (Chiodi, 2016), para a plataforma *AUTODESK 123D* .

Embora a classe de HAWT representada (grande porte) seja a mais utilizado, devido a sua elevada produção de energia, as HAWTs podem também ser turbinas eólicas de pequeno porte. Estas diferem entre si pelo diâmetro das pás do rotor D, a área varrida pelo vento A, a sua potência nominal P e a sua aplicação, sendo típicamente utilizadas para fins domésticos, em urbanizações, pequenas empresas e/ou em grandes urbanizações. A sua classificação de acordo com estes dados está representada na Tabela 2.1,(Costa et al., 2014).

Tabela 2.1: Classificação de turbinas eólicas de pequeno porte.

Sub-classe \Característica	**D [m]**	**A [m²]**	**P [kW]**	**Aplicações**
Micro-turbina	D <1,6	A <2	P <5	Domésticos
Mini-turbina	1,6 <D <5	2 <A <78,5	5<P <50	Urbanizações e pequenas empresas
Pequena-turbina	5 <D <16	78,5 <A <200	50 <P <500	Grandes urbanizações

2.3 Controlo de Suspensão, Amortecedor e Travão

Existem essencialmente três tipos de sistemas de suspensão, amortecedor e travões, que diferem entre si pelas suas componentes físicas e modos de funcionamento, impondo assim vantagens e desvantagens entre eles.

2.3.1 Sistema Passivo

Os sistemas de controlo passivo foram os primeiros tipos de controlo a surgir, sendo por isso, mais simples e de custo reduzido, comparativamente aos semi-ativos e ativos. Estes consistem em alterações das características físicas de um determinado sistema. Uma vez que os sistemas são alterados fisicamente, as suas componentes reagem de acordo com as forças, vibrações e deslocamentos que podem afetar o seu comportamento normal. Essas componentes, sendo tipicamente massas, amortecedores, suspensões de mola, entre outros, não produzem desgaste energético, pois nenhuma dessas componentes necessita de energia externa para funcionar. Esse fator implica maior segurança, porque mesmo numa circunstância de falha de energia não deixa de reagir.

Por outro lado, por serem componentes físicas, são dimensionadas com determinados coeficientes e parâmetros de atuação. Isso faz com que trabalhem numa gama de frequência muito reduzida e em circunstâncias inesperadas, não produzem os melhores resultados. Esta limitação suscitou a necessidade de um controlo ativo que pudesse trabalhar em diversas gamas de frequência, (Oliveira, 2015).

2.3.2 Sistema Ativo

Tal como nos sistemas de controlo passivo, os sistemas de controlo ativo reagem às perturbações dissipando a energia não desejada. Este tipo de controlo funciona com base nos dados captados por sensores de força, deslocamento, entre outros, e reage após o processamento desses dados. Para a interação com os sensores, estes sistemas necessitam de ser alimentados externamente, o que lhes permite, para além de dissipar energia não desejada, injetar energia no sistema que controlam.

Apesar da vantagem de estarem sempre ativos, receberem rápidamente os dados dos sensores, processarem rápidamente essa informação e fazerem com que o atuador reaja em tempo válido ao acontecimento, possuem diversas desvantagens, tais como: a necessidade de leis de controlo, sensores, atuadores e uma grande quantidade de energia para alimentar essas funcionalidades. Para além das desvantagens já expostas, existe também a questão de segurança, na medida em que todo o sistema necessita de uma alimentação externa para funcionar, ou seja, caso ocorra uma falha de energia todo o sistema deixa de funcionar, (Oliveira, 2015).

2.3.3 Sistema Semi-ativo

Tal como os sistemas de controlo passivo e ativo, o sistema de controlo semi-ativo reage às perturbações dissipando energia não desejada. Este possui características dos dois anteriores, possuindo assim componentes físicas, tais como o sistema de controlo passivo, mas em certas circunstâncias age como o sistema de controlo ativo através de um sinal de controlo, alterando algumas das suas propriedades em tempo real, mas nunca injetando energia no sistema. Por esse motivo, o consumo de energia é reduzido e, do ponto de vista de segurança não possui as falhas de um sistema de controlo ativo, mas sim as suas vantagens, (Oliveira, 2015).

Tipicamente, o sistema de controlo semi-ativo mais utilizado é o MR, embora exista uma abordagem idêntica nos ER, (Oliveira, 2015).

2.4 Tecnologia de fluídos Magnetoreológicos

Os fluídos Magnetoreológico (MR) são materiais que, quando expostos a um campo magnético, as suas partículas começam a alinhar-se de forma paralela ao sentido do campo magnético, (Oliveira, 2015). Esse alinhamento vai provocar alterações reversíveis nas propriedades reológicas do fluído, ou seja, na sua elasticidade, plasticidade e viscosidade, (Paré et al., 1998). A aplicação do fluído MR depende do seu design, (Dariush Ghorbany, 2011), que define os três possíveis modos de funcionamento:

- Modo de Válvula, Figura 2.10, normalmente utilizado como amortecedor, no qual a viscosidade do fluído varia de acordo com o campo magnético aplicado e, essa variação provoca uma diferença de pressão nos polos magnéticos. Neste modo, os polos magnéticos do sistema mantêm-se fixos, enquanto os fluídos são forçados a passar entre os mesmos, (Laboratory, 2016; Moura, 2003; Paré et al., 1998).

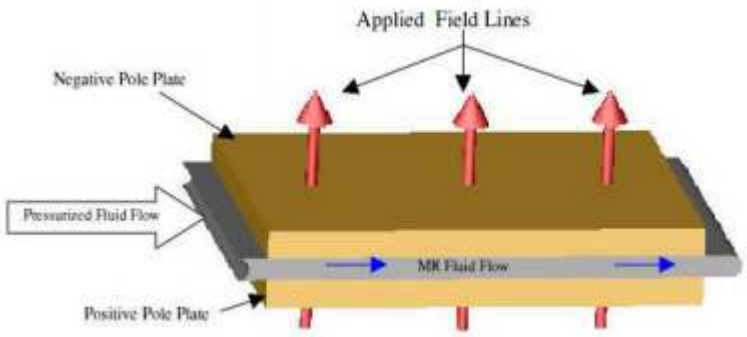


Figura 2.10: Sistema de fluído MR no modo de Válvula, (Dariush Ghorbany, 2011).

- Modo de Corte, Figura 2.11, utilizado em sistemas de travagem, onde uma das placas permanece fixa enquanto a outra está em movimento rotacional, (Dariush Ghorbany, 2011; Laboratory, 2016; Moura, 2003).

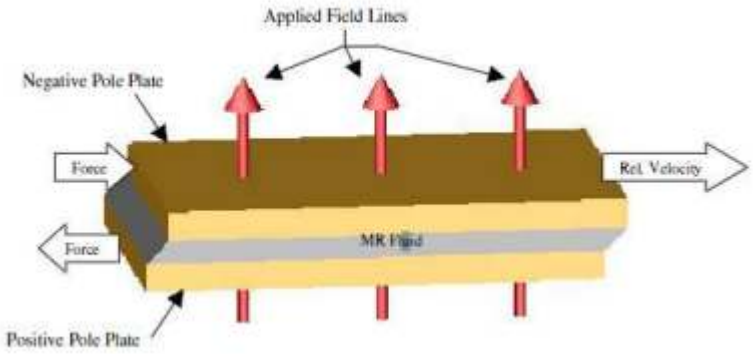


Figura 2.11: Sistema de fluído MR no modo de Corte, (Dariush Ghorbany, 2011).

- Modo de Aperto (*Squeeze-film*), Figura 2.12, utilizado para o controlo de pequenos movimentos, baseia-se na compressão dos dois polos magnéticos, criando um filme fino do fluído MR, (Dariush Ghorbany, 2011; Moura, 2003; Oliveira, 2015).

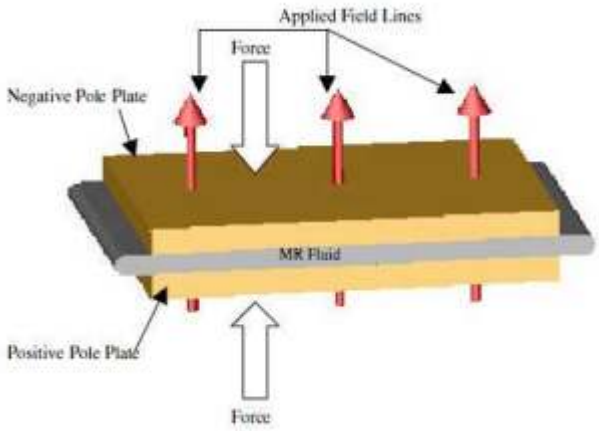


Figura 2.12: Sistema de fluído MR no modo de Aperto, (Dariush Ghorbany, 2011).

Com um comportamento e funcionamento idêntico a este sistema existem os fluídos ER (Eletro-Reológicos) que, segundo (Moura, 2003), apesar de serem idênticos, os MR possuem a vantagem de serem mais fortes, estáveis e fáceis de utilizar. Segundo (Paré et al., 1998), Carlson et al apontou mais vantagens do ponto de vista do *Yield Stress* total e da energia necessária para o seu funcionamento. Os ER possuem pequenas variações reológicas e também algumas variações com a temperatura, o que implica que para obter a mesma potência, deve ser aplicada uma elevada tensão elétrica no mesmo, enquanto que o MR necessita de pequenas correntes elétricas. Por outras palavras, o conceito é o mesmo,

trata-se de fluídos cuja viscosidade pode ser controlada aplicando um campo elétrico no caso dos ER ou, um campo magnético do caso dos MR, (Dariush Ghorbany, 2011).

Para se tirar melhor proveito do campo magnético, este deve ser aplicado de forma perpendicular ao fluxo do fluído, (Paré et al., 1998), uma vez que a intensidade do campo magnético produzido nas bobinas define as características reológicas do fluído, (Paschoal, 2011). É importante referir que este sistema pode ser configurado para operar de forma axial ou em ambientes rotacionais.

2.5 Travão Magnetoreológico

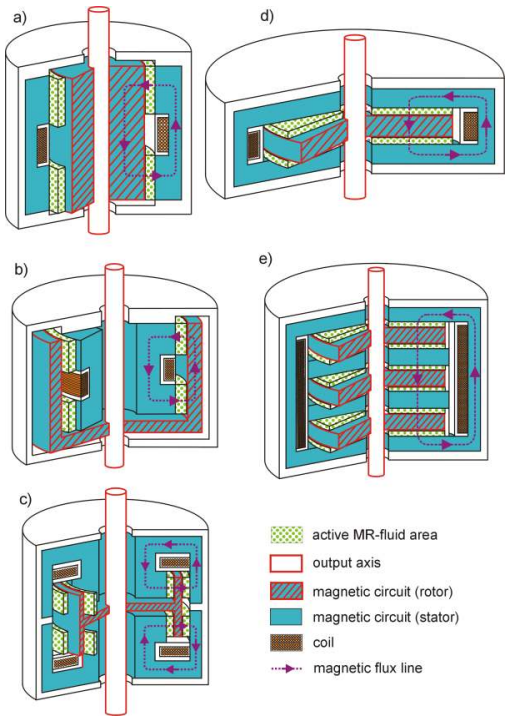


Figura 2.13: Tipos de travões MR, (Poznić et al., 2012).

O travão MR é realizado com o sistema de fluídos MR a funcionar no modo de corte, o que implica que um dos polos estará em movimento, quer seja de translação como de rotação. O tipo de travão varia com as suas características físicas, que altera a sua complexidade mecânica e os resultados produzidos, Figura 2.13. Segundo (Laboratory, 2016; Poznić et al., 2012), **a)** representa o travão *Drum*, **b)** representa o travão *Drum* invertido, **c)** o *T-shaped* rotor, **d)** o travão de disco e **e)** o travão de múltiplos discos.

Ao ser gerado um campo magnético, este fecha o seu circuito entre o rotor e o estator à volta da bobina. Como as partículas do fluído MR na presença de um campo magnético provocam variação de viscosidade, torna-se possível alterá-la de tal modo que provoque um amortecimento no polo em rotação . Quando esse amortecimento for acentuado é possível provocar a travagem de um objeto.

2.6 Sistemas de Caixa-Preta, Caixa-Cinzenta e Caixa-Branca

Os sistemas de caixa preta são os mais simples de utilizar, porque dependem apenas de dados de entrada e de saída. Nestes sistemas, o comportamento interno é definido pela ferramenta utilizada, de forma a que os resultados dependam dos dados de entrada introduzidos. Por este motivo, tornam-se mais simples e rápidos, visto que a ferramenta não necessita de identificar todos os passos processados internamente. Por outro lado, tornam os resultados questionáveis, tendo em conta que o utilizador não sabe o que se passa internamente e as aproximações realizadas podem não corresponder à realidade, (REDSTONE, 2008). Esta desvantagem desaparece quando é utilizado um sistema de caixa-branca, que permite observar o comportamento interno de um determinado sistema. Estes sistemas são mais complexos de implementar, mas garantem estabilidade e fornecem resultados precisos do ponto de vista interno e externo, (REDSTONE, 2008).

O sistema de caixa cinzenta é utilizado quando são conhecidos apenas alguns parâmetros e, os restantes são descobertos com base em algoritmos de identificação. Por outras palavras, trata-se de uma junção dos sistemas de caixa preta e de caixa branca.

2.7 Espaço de Estados

Utilizando os métodos clássicos da teoria de controlo, é possível descrever uma instalação LTI (Linear time-invariant), com base em funções de transferência que apenas relacionam a entrada e a saída da instalação, (Rowell, 2002). Quando é considerado um sistema MIMO (Multiple-Input-Multiple-Output), o mesmo tipo de processo pode ser realizado, descrevendo a instalação como sendo uma matriz composta por funções de transferência, (Paulo Gil, 2015), o que mais uma vez, não considera a dinâmica interna do sistema.

Este facto serve como justificação para a representação de um sistema em Espaço de Estados, na medida em que esta abordagem permite considerar a dinâmica interna do sistema, descrevendo-o como um conjunto de equações diferenciais de primeira ordem, no caso de sistemas SISO (Sigle-Input-Single-Output) e ordem superior no caso dos sistemas MIMO.

O Espaço de Estados é composto por um conjunto de variáveis de estados $\dot{x}$, uma matriz de estado A, uma matriz de entrada B, uma matriz de saída C e uma matriz de avanço D, sendo o sistema representado pelo diagrama de blocos da Figura 2.14, a sua equação de estados, Equação 2.4, e a sua equação de saída, Equação 2.5, (Paulo Gil, 2015; Rowell, 2002).

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.4}$$

$$Y(t) = Cx(t) + Du(t) \tag{2.5}$$

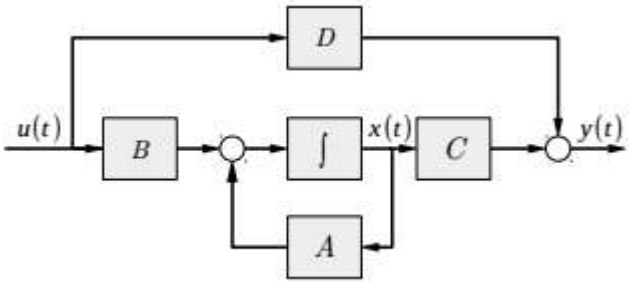


Figura 2.14: Diagrama de blocos do sistema em espaços de estados.

2.8 Controlo Difuso

O controlo Difuso foi desenvolvido por Loffi Zadeh, em 1965, com o propósito de manipular dados quando estes fossem subjectivos. Este pode ser utilizado em circunstâncias em que seja necessária a manipulação de informação aparentemente irrelevante, de forma a retirar uma conclusão, mesmo em situações não lineares, (Oliveira, 2015).

2.8.1 Difusificação

O controlador difuso baseia-se num conjunto de regras difusas do tipo :

```
1 Se (X) então (Y)
```

Baseia-se em três processos distintos: Difusificação, Inferência Difusa e Desdifusificação (Figura 2.15).

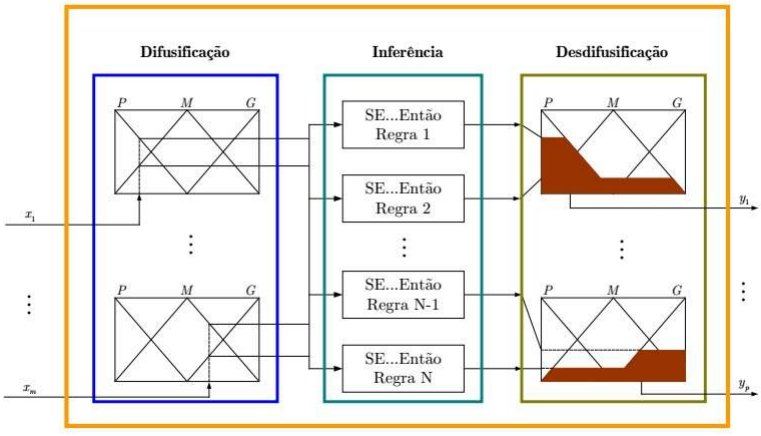


Figura 2.15: Arquitetura de um sistema de controlo difuso.

Na Difusificação, as entradas são representadas pelas funções de pertença e as suas respectivas variáveis linguísticas. As variáveis linguísticas de entrada descrevem X e as variáveis linguísticas de saída descrevem Y, sendo que entre a entrada e a saída situa-se a inferência, (Oliveira, 2015; Paschoal, 2011).

Um exemplo de variáveis linguísticas pode ser o seguinte:

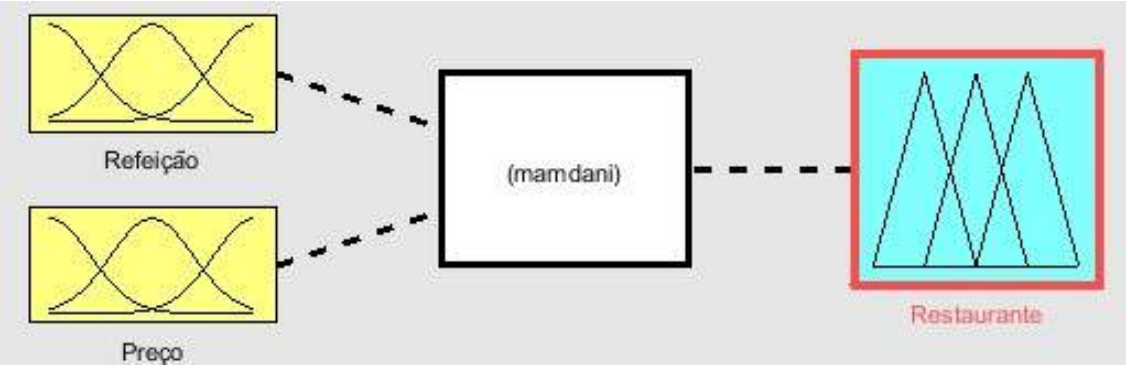


Figura 2.16: Controlo difuso - Definição das variáveis de entrada e de saída.

Após um jantar num determinado restaurante, um indivíduo avaliou a qualidade da refeição e o preço, a fim de determinar a qualidade do restaurante. A partir desta descrição é possível definir as duas variáveis de entrada e a variável de saída (Figura 2.16).

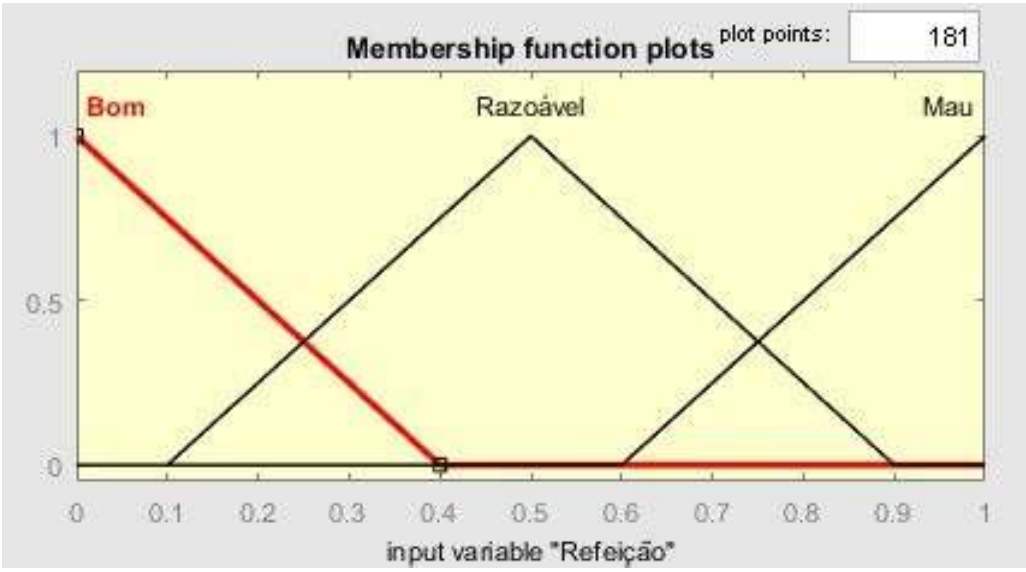


Figura 2.17: Controlo difuso - Variáveis linguísticas.

Os parâmetros avaliados foram: o jantar foi bom, razoável ou mau, barato ou caro. Dependendo da relação entre a qualidade da refeição e o preço, é possível concluir se foi um bom, ou um mau restaurante. Todas estas definições representam variáveis linguísticas que descrevem tanto as entradas como a saídas.

A Figura 2.17 demonstra o exemplo de variáveis linguísticas definidas para a variável de entrada "Refeição", embora estas possam ser de diversos tipos de funções de pertença (Figura 2.18).

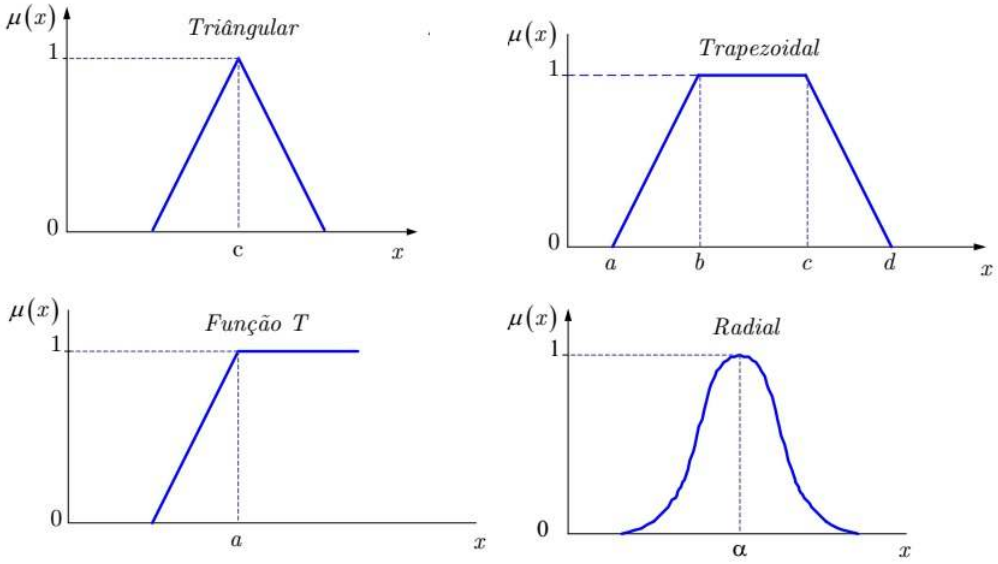


Figura 2.18: Controlo difuso - Funções de Pertença.

2.8.2 Inferência Difusa

É nesta etapa que todo o tratamento das regras é realizado, ou seja, a forma como são interpretadas as regras. Para a inferência, existem diversos modelos clássicos como o modelo de Mamdani e o modelo de Takagi-Sugeno, sendo que cada um serve um propósito distinto, (Oliveira, 2015; Paschoal, 2011).

2.8.2.1 Inferência Difusa: Takagi-Sugeno

Este modelo de inferência é apresentado por Tomohiro Takagi e Michio Sugeno como uma ferramenta para a criação de modelos com base nas regras difusas. Descrevendo as entradas como um conjunto de modelos lineares, foi possível a representação de um modelo não linear (Figura 2.19), (Takagi e Sugeno, 1985).

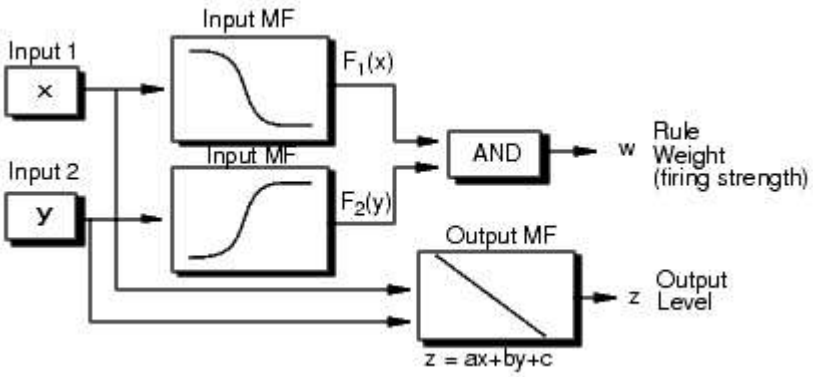


Figura 2.19: Funcionamento do modelo Takagi-Sugeno, (MathWorks, 2016).

2.8.2.2 Inferência Difusa: Mamdani

As implicações do modelo de Mamdani baseiam-se na teoria clássica dos conjuntos e nas regras de União (Agregação) e Intersecção (Mínimo), Figura 2.20, Equações 2.6 e 2.7 .

$$(A \wedge B) \Leftrightarrow (\mu_A \wedge \mu_B) \Rightarrow min(\mu_A, \mu_B) \tag{2.6}$$

$$(A \vee B) \Leftrightarrow (\mu_A \vee \mu_B) \Rightarrow max(\mu_A, \mu_B) \tag{2.7}$$

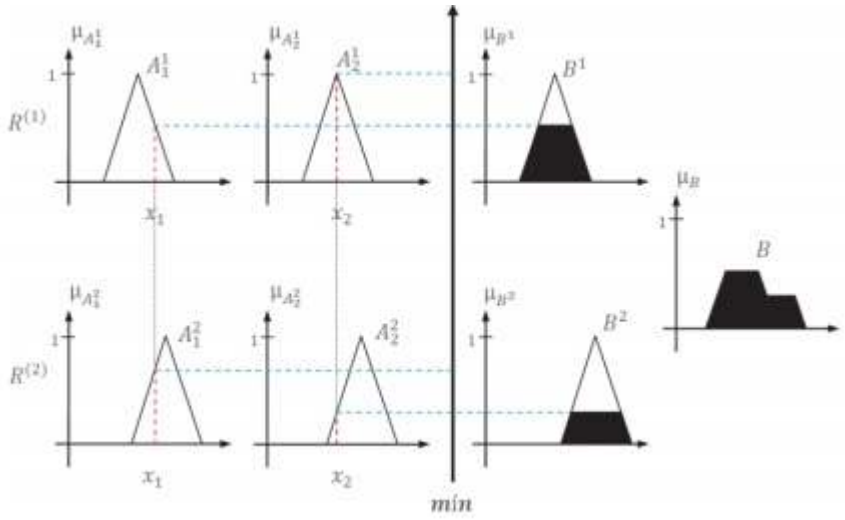


Figura 2.20: Mecanismo de Inferência de Mamdani, (Oliveira, 2015).

2.8.3 Desdifusificação

Após a Difusificação e a aplicação das regras pelo método de inferência, os dados são reconvertidos do conjunto difuso para valores reais. Esta conversão denomina-se desdifusificação e, para este processo existem vários métodos, como é o caso do método do centro da área, o método da altura e o método da média dos máximos.

2.9 Redes Neuronais

A rede neuronal é uma técnica computacional cujo funcionamento é inspirado no sistema nervoso animal. Em 1943, *McCulloch & Pitts* propuseram o primeiro modelo de um neurónio artificial e anos depois, foram introduzidos mecanismos de aprendizagem, algoritmo de treino baseado no método dos mínimos quadrados, algoritmo de retropropagação, entre outros.

Esta rede permite criar modelos e estimar a sua resposta (y) com base nas entradas do modelo (u_1 e u_2), pesos (W_i), polaridade (b), estados de ativação (x) e função de ativação (σ), Figura 2.21. Esta é composta por três camadas, Figura 2.22 :

- Camada de entrada (Entrada de dados) - os neurónios associados a esta camada dependem da natureza do sistema dinâmico;
- Camada interna - possui um número diversificado de neurónios;
- Camada de Saída (Saída de dados) - os neurónios associados a esta camada dependem da natureza do sistema dinâmico.

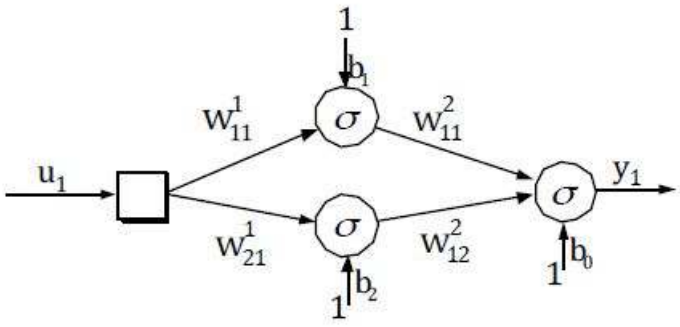


Figura 2.21: Componentes de uma rede neuronal .

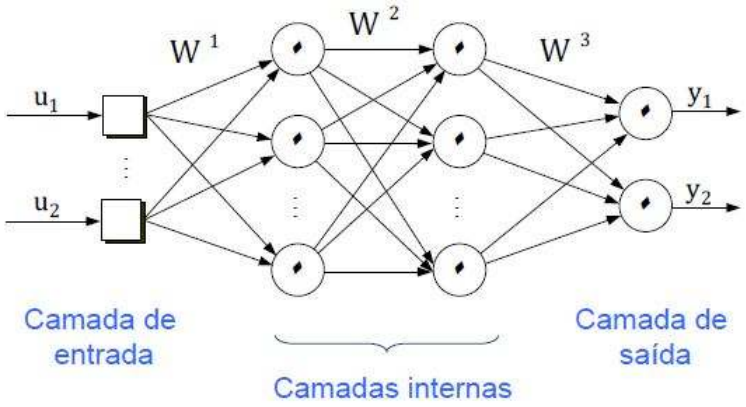


Figura 2.22: Estrutura das redes proativas multicamada .

De forma a incorporar informação temporal na estrutura da rede é utilizado o vetor de regressão, Equação 2.8, onde a saída atual depende das saídas e entradas anteriores.

$$y(k) = f(y(k-1),...,y(k-n_a,u(k-1),...,u(k-n_b)) \tag{2.8}$$

$y(k)$ é a resposta do modelo, n_a é o atraso associado à saída e n_b é o atraso associado à entrada. Caso seja introduzida pouca informação temporal as redes neuronais podem não conseguir representar o comportamento real desejado. Por outro lado, caso seja introduzida demasiada informação, o peso computacional aumenta e o preditor neuronal torna-se fortemente suscetível a efeitos indesejados, relacionados com o ruído.

A rede neuronal pode também ser treinada, de forma a minimizar o erro entre a saída da rede e a do sistema que se pretende modelar.
Este processo iterativo permite sucessivas correções até que a rede consiga representar fielmente o comportamento do sistema, (Gil, 2013).

Um dos problemas da rede neuronal está relacionado com o seu processo de treino, em que o erro torna-se tão reduzido que esta perde a generalização. Ou seja, a rede memoriza os exemplos presenciados durante o treino mas, quando são introduzidos valores diferentes dos conhecidos, esta não sabe como reagir terminando, tipicamente, com respostas oscilatórias. Existem diversos métodos para solucionar este problema, como por exemplo: *Retraining Neural Networks*, *Multiple Neural Networks*, *Early Stopping*, *Regularization*, *Scaled Conjugate Gradient*, (MathWorks, 2017b).

2.10 Filtro de Kalman

O Filtro de Kalman é um método recursivo que permite estimar e/ou filtrar o comportamento de um determinado sistema, minimizando o seu erro quadrático. Essa estimativa pode ser realizada para o passado, para o presente e para o futuro , sendo que após este processo, o ruído medido é minimizado. As equações que descrevem o Filtro de Kalman estão divididas em dois grupos, sendo :

$$\hat{x}_k = \hat{x}_{k-1} \tag{2.9}$$

$$\hat{P}_k = \hat{P}_{k-1} + Q \tag{2.10}$$

$$K_k = \frac{P_k}{P_k + R} \tag{2.11}$$

$$\hat{x}_k = \hat{x}_k + K_k(z_k - \hat{x}_k) \tag{2.12}$$

$$P_k = (1 - K_k)P_k \tag{2.13}$$

- Equações de *Time update*, responsáveis pelo avanço no tempo, Equações 2.9 e 2.10, onde $\hat{x}_k$ representa o valor estimado, $\hat{P}_k$ o erro de estimação e Q o erro aceitável;

- Equações de *Measurement update* responsáveis pela redução do erro, Equações 2.11, 2.12 e 2.13, onde K_k representra o ganho, R o erro de medição (covariância) e z_k o valor medido.

A lógica de implementação do algorítmo demonstra que o processo é iterativo e pode ser observada na Figura 2.23.

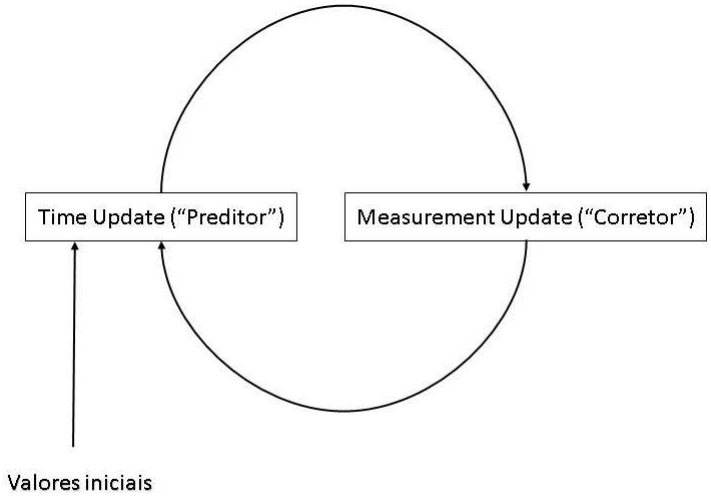


Figura 2.23: Funcionamento do Filtro de Kalman.

2.11 Arduíno Uno

O Arduíno Uno é um micro-controlador que permite uma fácil interação com o hardware e o software.
Permite a receção de dados, normalmente provenientes de sensores, reproduzindo a saída desejada ao ser introduzido um conjunto de instruções na placa do micro-controlador, Figura 2.24.

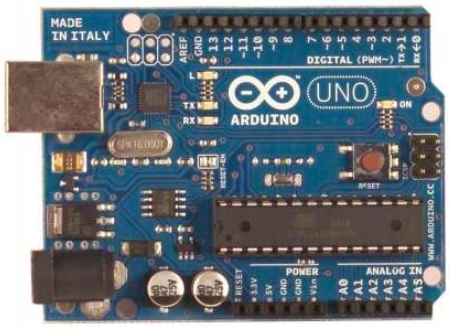


Figura 2.24: Arduíno Uno, (Melorose et al., 2015).

As instruções são baseadas nas linguagens de programação $C/C++$, permitindo a utilização de qualquer função existente nelas, (Arduino, 2016).

O Arduíno Uno utiliza o protocolo de comunicação I^2C para comunicar com os restantes hardwares. Este protocolo utiliza duas linhas de comunicação: SDA e SCL. Para que haja uma comunicação por I^2C é necessário que dois circuitos integrados tentem se comunicar, sendo esta relação chamada de *Master/Slave*, onde o *Master* é o que deseja comunicar e o *Slave* o segundo membro na comunicação. Ser Master pode significar *Master-transmitter* ou *Master-receiver* e, segundo (NXP Semiconductors, 2014), quando

existem dois ou mais Masters a iniciar uma transferência em simultâneo, a linha multi-master possui a capacidade de detetar colisões e corrupções nos dados, melhorando a qualidade da comunicação.

Este possui portas analógicas e digitais, em que as portas digitais 2 e 3 podem ser utilizadas pelos *Interrupts* e, as portas analógicas possuem um conversor com 10 bits de resolução que devolvem números inteiros de 0 a 1023, (Arduino, 2017).

2.12 Sensor MPU-6050

O sensor MPU-6050 é um sensor que contém um giroscópio e um acelerómetro com a capacidade de medição nos diversos eixos X, Y e Z, Figura 2.25.

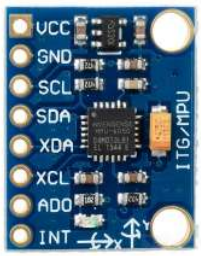


Figura 2.25: Sensor MPU-6050.

O acelerómetro e o giroscópio são aparelhos de medição de posição/velocidade angular e aceleração, que neste caso está associado ao sensor MPU-6050 e comunica com o Arduíno através do protocolo I^2C. No *datasheet* do sensor, é explicado o seu comportamento relativamente à sua sensibilidade e fator de escala, ou seja, é possível definir a escala de medição do mesmo alterando o seu fator de escala com base em funções definidas na biblioteca do MPU-6050. Ao ser alterado o fator de escala, a precisão de medição do sensor é alterada e, por sua vez, o fator de sensibilidade. Para este sensor, as opções de sensibilidade descritas na Figura 2.26 e Figura 2.27, são constantes que dividem os valores obtidos do sensor, de modo a que se obtenham dados reais.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
GYROSCOPE SENSITIVITY					
Full-Scale Range	FS_SEL=0		±250		°/s
	FS_SEL=1		±500		°/s
	FS_SEL=2		±1000		°/s
	FS_SEL=3		±2000		°/s
Gyroscope ADC Word Length			16		bits
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)
	FS_SEL=1		65.5		LSB/(°/s)
	FS_SEL=2		32.8		LSB/(°/s)
	FS_SEL=3		16.4		LSB/(°/s)
Sensitivity Scale Factor Tolerance	25°C	-3		+3	%
Sensitivity Scale Factor Variation Over Temperature			±2		%
Nonlinearity	Best fit straight line; 25°C		0.2		%
Cross-Axis Sensitivity			±2		%

Figura 2.26: Dados relativos ao Giroscópio, (Ave, 2013).

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
ACCELEROMETER SENSITIVITY					
Full-Scale Range	AFS_SEL=0		±2		*g*
	AFS_SEL=1		±4		*g*
	AFS_SEL=2		±8		*g*
	AFS_SEL=3		±16		*g*
ADC Word Length	Output in two's complement format		16		bits
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*
	AFS_SEL=1		8,192		LSB/*g*
	AFS_SEL=2		4,096		LSB/*g*
	AFS_SEL=3		2,048		LSB/*g*
Initial Calibration Tolerance			±3		%
Sensitivity Change vs. Temperature	AFS_SEL=0, -40°C to +85°C		±0.02		%/°C
Nonlinearity	Best Fit Straight Line		0.5		%
Cross-Axis Sensitivity			±2		%

Figura 2.27: Dados relativos ao Acelerómetro, (Ave, 2013).

2.13 Sensor AltIMU-10 v4

O sensor AltIMU-10 v4, Figura 2.28, é um IMU (Inertial Measurement Unit) e um Altímetro que contém um giroscópio, um acelerómetro, um magnetómetro e um barómetro digital, sendo portanto um sensor que possui a capacidade de medir a pressão, a rotação, a aceleração e o magnetismo de forma a conseguir calcular a sua orientação segundo os eixos de *Yaw*, de *Pitch*, de *Roll* e da altitude.

Este possui cinco pins:

- SDA - *Serial Data Line*, responsável pela transmissão de dados utilizando o protocolo de comunicação I^2C;
- SCL - *Serial Clock Line*, utilizado pelo mesmo protocolo;
- VIN - Pin pelo qual o sensor é alimentado elétricamente, $[2.6 - 5.5]V$;
- GND - Ground, 0V;
- VDD - Serve para alimentar algum dispositivo, caso seja necessário.

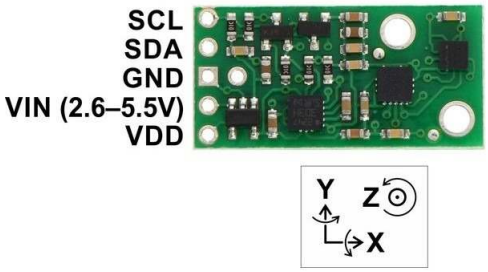


Figura 2.28: AltIMU-10 v4, (Pololu, 2017).

Quando utilizado em conjunto com o Arduíno Uno, este sensor utiliza três bibliotecas, "3G", "LSM303" e "LPS", sendo que a "L3G" serve para facilitar a leitura dos dados do giroscópio, o "LSM303" facilita a interação e leitura dos dados do acelerómetro e do magnetómetro e, a biblioteca "LPS" simplifica a leitura dos dados de pressão.

Tal com o MPU-6050, este sensor também possui factor de escala e de sensibilidade que permitem escolher a precisão desejada, representados nas Tabelas 2.29 e 2.30, (Pololu, 2017).

Symbol	Parameter	Test conditions	Min.	Typ.(1)	Max.	Unit
LA_FS	Linear acceleration measurement range(2)			±2		*g*
				±4		
				±6		
				±8		
				±16		
M_FS	Magnetic measurement range			±2		gauss
				±4		
				±8		
				±12		
LA_So	Linear acceleration sensitivity	Linear acceleration FS=±2g		0.061		mg/LSB
		Linear acceleration FS=±4g		0.122		
		Linear acceleration FS=±6g		0.183		
		Linear acceleration FS=±8g		0.244		
		Linear acceleration FS=±16g		0.732		
M_GN	Magnetic sensitivity	Magnetic FS=±2gauss		0.080		mgauss/ LSB
		Magnetic FS=±4gauss		0.160		
		Magnetic FS=±8gauss		0.320		
		Magnetic FS=±12gauss		0.479		

Figura 2.29: Dados relativos ao Acelerómetro e ao Magnetómetro, (STMicroeletronics, 2012).

Symbol	Parameter	Test condition	Min.	Typ.(1)	Max.	Unit
FS	Measurement range	User selectable		±245 ±500 ±2000		dps
So	Sensitivity			8.75 17.50 70.00		mdps/digit

Figura 2.30: Dados relativos ao Giroscópio, (STMicroeletronics, 2013).

2.14 Módulo RF

Os Módulos RF são dispositivos eletrónicos que servem para comunicação wireless através de rádio frequência.

Existem três tipos de módulos RF:

- *Transmitter* - Transmissor, serve para enviar dados tanto para o *Receiver* como para o *Transceiver*.
- *Receiver* - Recetor, tem como propósito receber os dados enviados tanto pelo *Transmitter* como pelo *Transceiver*.
- *Transceiver* - Funciona como uma junção do *Transmitter* e do *Receiver*, permitindo enviar e receber dados.

Normalmente, são acoplados a um Arduíno ou Raspberry pi de forma a ser controlada a troca de dados.

2.14.1 Módulo RF 433Mhz AM

Este módulo RF, Figura 2.31, possui um transmissor e um recetor com as especificações representadas nas Tabelas 2.2 e 2.3.

Figura 2.31: Módulo RF 433Mhz, (ELECTROFUN, 2016).

Tabela 2.2: Especificação do Módulo RF Transmissor

Transmissor	
Modelo	MX-FS-03V
Alcance	20-200m
Tensão elétrica de operação	3.5-12V
Modo de operação	AM
Taxa de transferência	4KB/s
Potênciade Transmissão	10mW
Frequênciade transmissão	433MHz
Pin	Dados-VCC-GND
Dimensões	19x19mm

Tabela 2.3: Especificação do Módulo RF Recetor

Recetor	
Modelo	MX-05V
Tensão elétrica de operação	5V DC
Corrente elétrica de operação	4mA
Frequência de receção	433Mhz
Sensibilidade	-105dB
Dimensões	30 x14 x 7 mm

Como se pode observar pelas especificações, este módulo RF funciona a uma frequência fixa de $433Mhz$ e possui um alcance máximo dependente da tensão elétrica aplicada de $200m$, como é esperado de um sistema a utilizar a modulação AM.

Os pins do emissor e do recetor são identificados na Figura 2.32, e o esquema de ligação ao Arduíno Uno nas Figuras 2.33 e 2.34. Para a programação e controlo destes módulos através do Arduíno Uno é aconselhado o uso da biblioteca "VirtualWire", tanto para a transmissão como para a receção, (PJRC, 2016).

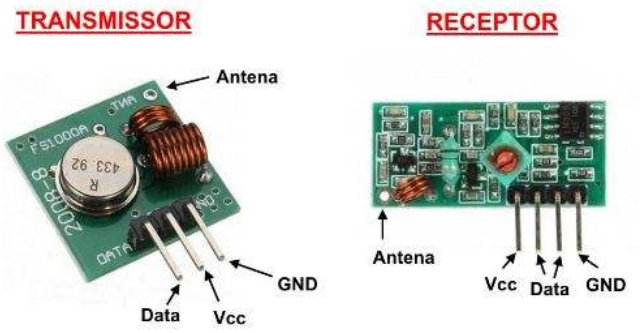


Figura 2.32: Identificação dos pins do módulo RF 433Mhz, (FILIPEFLOP, 2016).

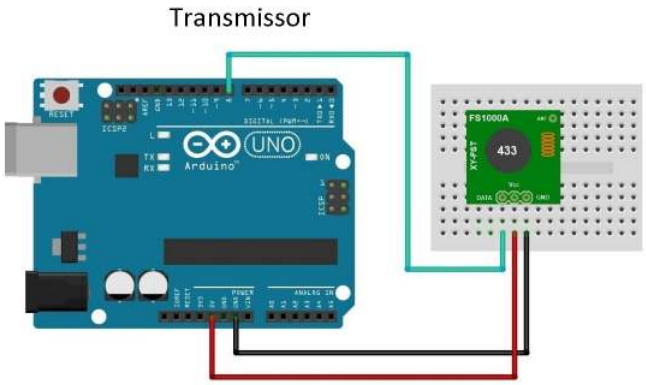


Figura 2.33: Esquema de ligação do módulo RF 433Mhz transmissor ao Arduíno Uno, adaptação de (FILIPEFLOP, 2016).

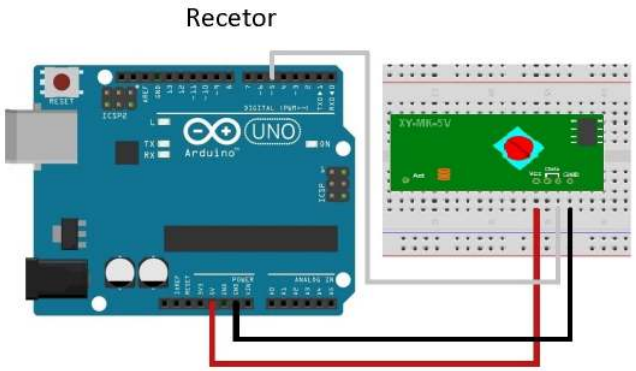


Figura 2.34: Esquema de ligação do módulo RF 433Mhz recetor ao Arduíno Uno, adaptação de (FILIPEFLOP, 2016).

2.14.2 Módulo RF APC220

O módulo RF APC220, Figura 2.35, é um módulo de comunicação wireless *Semi-duplex* em rádio frequência que funciona como transmissor e como recetor, sendo portanto um *Transceiver*. A comunicação é realizada através de uma porta *Serial* de um Arduíno Uno, de um Raspberry pi ou através do UART/TTL.

Este possui as caraterísticas descritas nas Tabelas 2.4 e 2.5, onde se pode verificar a possibilidade de definir a frequência de funcionamento, o ritmo das portas RX (porta de receção) e TX (porta de transmissão) e, as características de rede. Também é possível observar um aumento significativo no alcance, na diferença na modulação e na tensão de alimentação, entre outros parâmetros, que demonstram a superioridade do módulo RF APC220 face ao módulo RF 433*Mhz*.

A identificação dos pins e o esquema de ligação deste módulo estão representados nas Figuras 2.36 e 2.37 respetivamente, sendo o pin "SET" ligado ao "VCC" de forma a defini-lo como "HIGH". Esta operação permite ao módulo estar no modo de configuração.

Figura 2.35: Módulo RF APC220.

Tabela 2.4: Especificação do Módulo RF APC220, (PTROBOTICS, 2016).

APC220	
Alcance	1000m a 9600 bps
Buffer	256 bytes
Sensibilidade	-112 dbm a 9600 bps
Modulação	GFSK
Interface	UART/TTL
Dimensões	37 x 17 x 6.5
Conversor USB para TTL	CP210

Tabela 2.5: Especificações variáveis do Módulo RF APC220, (DFROBOT, 2016)

Paramêtro	**Intervalo de valores**	**Definição Padrão**
Frequência RF	Resolução 1Khz, Precisão +-100Hz	434Mhz
Ritmo TRx	1200, 2400, 4800, 9600, 19200 bps	9600 bps
VCC	3.3-5.5V	5V
Series Rate	1200,2400,4800,9600,19200,38400,57600 bps	9600 bps
NET ID	0-65535 (16 bit)	12345
NODE ID	123456789012	
Series Parity	Disable, Odd Parity, Even Parity	Disable

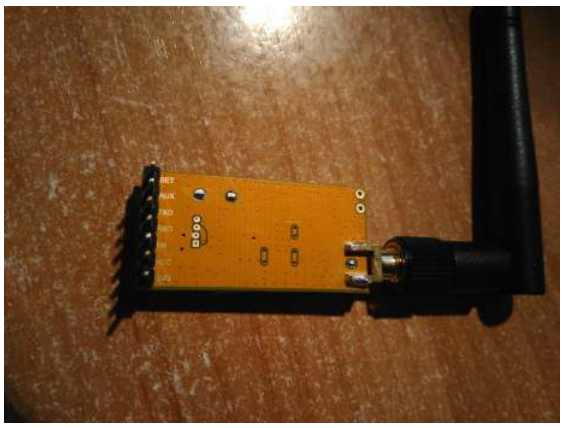


Figura 2.36: Pins do módulo RF APC220,(DFROBOT, 2016).

- SET - Set parameters (low);
- AUX - UART Signal- Receive (low) Transmit (high);
- TXD - UART TX;
- RXD - UART RX;
- EN - Disable the device when apply <0.5V, Enable the device when leave it disconnected or apply >1.6V;
- VCC - 3.3V-5.5V Power;
- GND - 0V Ground.

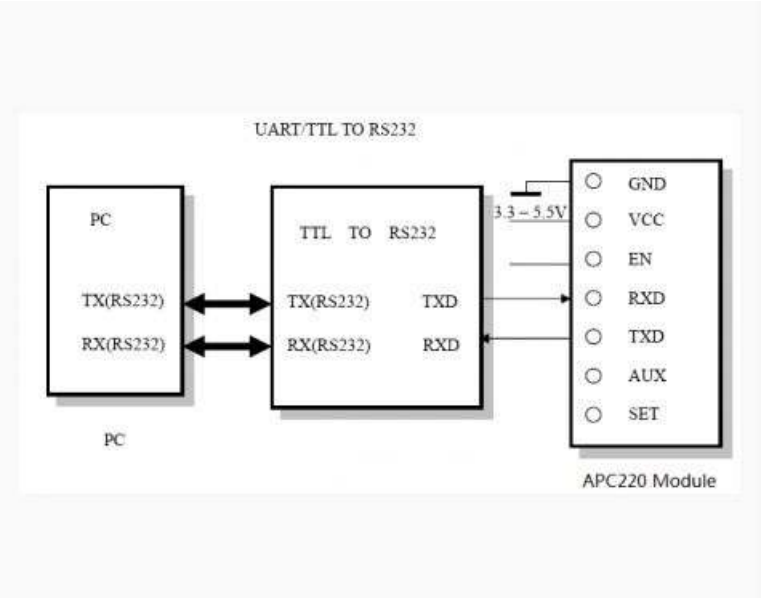


Figura 2.37: Esquema de ligação do módulo RF APC220 ao PC através do UART/TTL, (DFROBOT, 2016).

2.15 Encoder Incremental

Os Encoders são aparelhos que permitem a medição da velocidade, da posição e da aceleração angular. A Figura 2.38 representa o *Eltra EL-ER 58C*, onde se pode observar este Encoder em duas perspetivas diferentes. Na **a)** estão representados os três pontos de referência da posição angular. O Eixo de rotação, representado na **b)**, ao começar a rodar passa pelas posições indicadas em **a)** contando 120° o intervalo completo entre cada ponto. Desta forma é possível contabilizar os 360° que correspondem a uma Rotação/Revolução (300 pulsos).

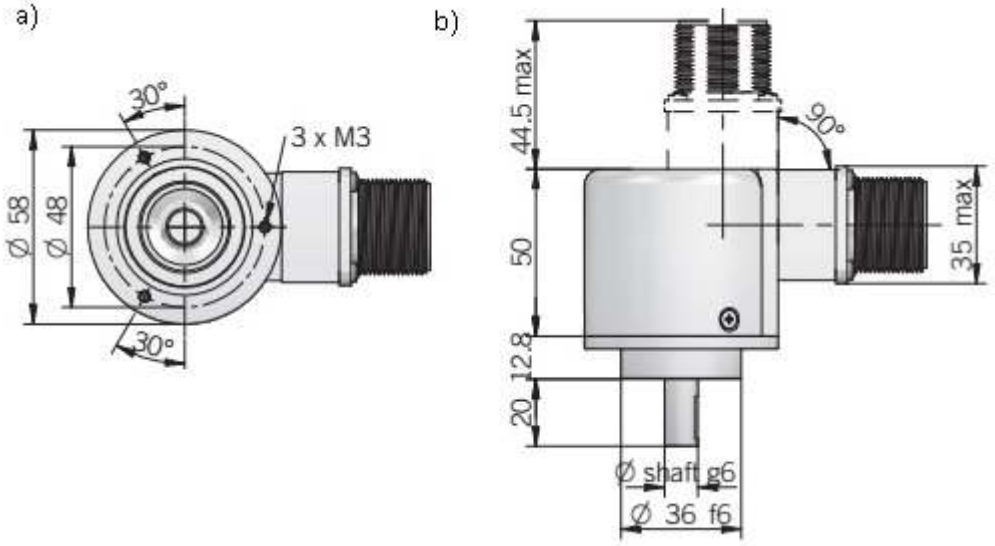


Figura 2.38: Eltra EL-ER 58C,(Eltra, 2014).

A posição angular medida pelo Encoder tem como unidade *Pulse/Rev*, que indica quantos pulsos serão contados ao fim de uma rotação completa. Este valor é fornecido pelo fabricante como sendo a sensibilidade do Encoder e serve de referência, pois permite o cálculo da posição angular em qualquer instante.

Se o Encoder possuísse um único conjunto de pulsos, não seria útil, (National Instruments, 2016). Normalmente, este possui dois canais de saída, o canal A e o canal B, que devolvem pulsos desfasados de 90°. Caso os pulsos do canal A estejam desfasados em relação aos pulsos do canal B, existe um determinado sentido de rotação e na situação oposta, outro sentido de rotação.

Segundo (National Instruments, 2016), para se obter uma certa precisão nas medições com o Encoder, existem pelo menos três tipos de codificações : $X1$ (Figura 2.39), $X2$ (Figura 2.40) e $X4$ (Figura 2.41).

Na codificação $X1$, quando o pulso do canal A está à frente do pulso do canal B a contagem dos pulsos é incrementada. Caso contrário é decrementada, ou seja, de 180° em 180°.

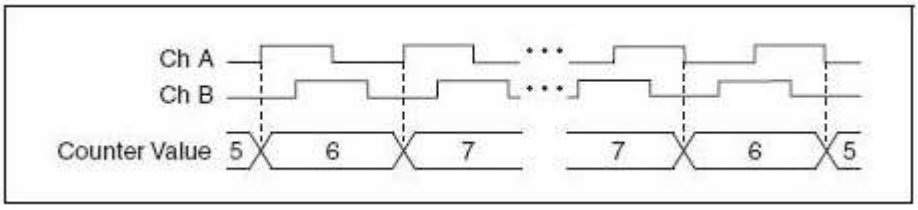


Figura 2.39: Codificação em X1, (National Instruments, 2016).

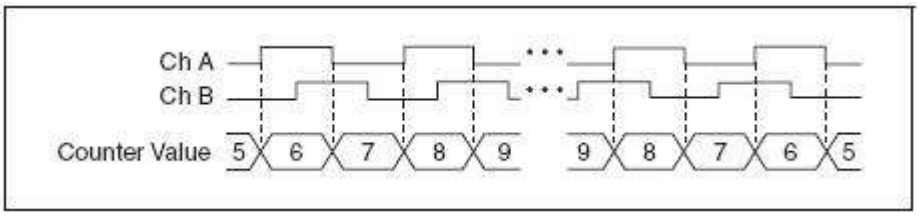


Figura 2.40: Codificação em X2, (National Instruments, 2016).

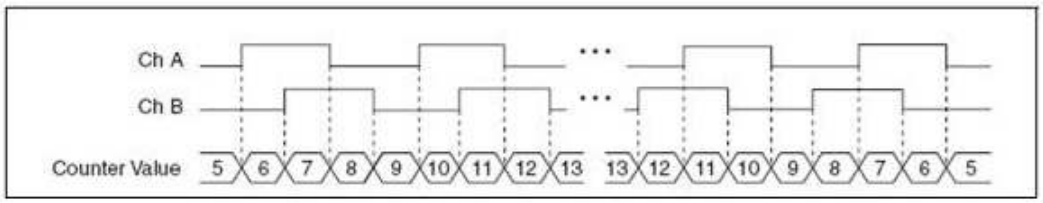


Figura 2.41: Codificação em X4, (National Instruments, 2016).

Na codificação $X2$, o contador é incrementado e decrementado todas as vezes que um pulso se tornar **HIGH** e todas as vezes que um pulso se tornar **LOW**, de 90° em 90°, sendo que nos últimos 90° ambos os pulsos estão em **LOW** e não há incremento nem decremento.

Na codificação X4 o contador de pulsos só é incrementado ou decrementado dependendo do pulso que está a frente.

2.16 Trabalhos relacionados

Atualmente, no que diz respeito à energia eólica, devido à dependência da velocidade, da direção do vento e de outros parâmetros, como a pressão e a temperatura, diversos estudos têm sido realizados de forma a estimar a energia elétrica produzida.

(Vladislavleva et al., 2013), implementou um preditor de energia eólica produzida em pequena escala, com base nos dados obtidos de uma estação meteorológica. Após uma análise comparativa com os dados da produção real pôde concluir que a estimativa era bastante aproximada, no entanto apenas previa 80% à 85% da produção real.

(Buffard et al., 2014), demonstrou que o método *K-nearest neighbors* possui uma boa precisão, quando utilizado para estimar os dados de produção de energia eólica em situações em que existam falhas. Este método permite detetar falhas que possam influenciar negativamente a performance deste sistema, como por exemplo um erro na medição da velocidade do vento por parte da *Nacelle*.

Estes estudos demonstram o interesse em compreender as diferenças entre produção real de um aerogerador, relativamente ao que poderia ser a sua produção caso este considerasse com maior precisão os dados da estação meteorológica. É importante considerar estes casos, pois poderiam ser aplicados no presente estudo, de forma a ser estimada uma produção elétrica com base nos diversos fatores que a compõem (dados meteorológicos) e, esta ser comparada à produção elétrica ideal estimada. Quando estimada a produção elétrica em condições ideais esta tende a ser superior à produção real. No entanto, a produção elétrica real pode ser maximizada quando usados métodos de controlo apropriados, normalmente aplicados aos ângulos de *Pitch* e de *Yaw*.

Foram utilizadores inicialmente controladores como o PID, devido à sua simplicidade, mas o mesmo necessita de conhecer a resposta do sistema, de forma a serem retirados os parâmetros para o cálculo dos ganhos. Por outro lado, por ser um SISO, seria complicado controlar mais do que uma saída, a não ser que fosse aumentada a sua complexidade, (Stol, 2002).

Com o aumento da produção destes sistemas começou-se a verificar problemas com a interação das turbinas. (Gebraad et al., 2014), afirmou que existindo mais que um aerogerador, um deles irá influenciar a quantidade de energia produzida, as cargas assíncronas sofridas, a turbulência do ar e a redução de velocidade angular do outro. A solução foi controlar o ângulo de *Yaw* de forma a definir um ponto 'ótimo' de operação para todos.

Numa outra abordagem, (Zhao e Stol, 2007) demonstraram que, controlando o ângulo de Pitch de cada uma das pás de forma individual, utilizando o *Periodic State-space* para esse efeito, era possível um controlo activo do *Yaw*. Um outro objetivo era a diminuição das cargas de torsão provocada pelas constantes rotações da turbina. Normalmente, são utilizados motores de *Yaw Drive*, o que implica a inserção de um sistema de *Yaw Drive* e uma resposta mais lenta. Mas (Zhao e Stol, 2007), concluíram que era possível a diminuição das cargas de torção, quando retirado o sistema de *Yaw Drive*, ou seja, os sistemas cujos movimentos no ângulo de *Yaw* sejam *free-yaw* .

Estas abordagens demonstram o interesse em otimizar os controladores para o sistema de produção de energia eólica. Com o aparecimento de tecnologias de controlo ativo e semi-ativo de suspensões e amortecedores, alguns sistemas de controlo começaram a ser adaptados e aplicados maioritariamente em veículos.

(Moura, 2003), realizou um estudo sobre um dos mais utilizados sistemas de suspensão semi-ativa aplicado em veículos, baseado no MRF. Para além da explicação teórica do seu funcionamento, implementou também modelos matemáticos que representam o sistema baseado nos modelos de $\frac{1}{4}$, $\frac{1}{2}$ e no modelo completo do veículo. Por fim, realizou uma análise comparativa dos resultados entre os sistemas de suspensão passiva, ativa e semi-ativa. Verificou-se as vantagens do sistema de suspensão semi-ativa comparativamente às outras, na medida em que o custo, o desempenho e as características do sistema de suspensão ativa são mais eficientes, porém exigem uma maior quantidade de energia durante o seu funcionamento.

Informações relativas aos modelos e diferenças entre os sistemas podem ser também encontrados em (Darus, 2008), onde foram obtidos modelos matemáticos que representassem um sistema ativo e passivo para um modelo completo de um veículo, baseando-se inicialmente no modelo de um $\frac{1}{4}$ de veículo. Apesar de considerar um controlador difuso entre outros controladores, utilizou o controlador *LQG* para o controlo da suspensão, tendo como objetivo o conforto dentro do veículo. Como conclusão, realizou uma comparação entre os dois tipos de suspensões de forma a ser clara a melhoria de resultados, aquando a utilização da suspensão ativa.

(Shuqin, 2011), estudou o comportamento de uma *VAWT* do tipo *Darrieus* quando é utilizado um sistema de suspensão ativa magnética, com o objetivo de reduzir o atrito nos rolamentos durante o seu movimento, no arranque e no *Self-pitch* suspenso magneticamente. Foi aplicado numa *VAWT*, devido ao baixo custo de implementação, a estrutura simplificada das pás, a instalação conveniente em áreas urbanas, a possibilidade de aproveitar o vento em todas as direções sem a necessidade de um mecanismo de direção e a possibilidade de poder ser instalada nos telhados sem o problema do ruído, ou seja, as vantagens da *VAWT* em comparação à *HAWT*. Com a utilização dos rolamentos magnéticos utilizados foi possível observar uma redução do atrito e do ruído.

(Lajqi e Pehan, 2012), explicaram os procedimentos para o design de um sistema não linear de suspensão ativa e semi-ativa. Partindo do modelo de $\frac{1}{4}$ de veículo explicaram a diferença entre os sistemas passivo, ativo e semi-ativo, utilizando o modelo de *Skyhook*. Para a simulação e observação dos resultados foi utilizado um controlador ON-OFF, que dependendo das vibrações transmitidas pelo piso ao pneu, este aplica uma determinada força de forma a manter o veículo estável.

Tendo em conta que um dos objetivos desta dissertação é a utilização de um sistema de controlo semi-ativo para travar o aerogerador na direção do vento, importa estudar a sua aplicação em objetos com deslocamento e velocidade angular.

(Park et al., 2006), na sua investigação, propôs e implementou um travão MR de Discos com dois discos, que iria trazer diversas vantagens relativamente aos sistemas de travões

hidráulicos. Sabendo que o travão de disco utiliza o fluído MR, que este fluído varia o seu *Yield Stress*(Tensão de escoamento) em função do campo magnético a que é exposto e que o campo magnético, varia com a variação de corrente elétrica aplicada, então o travão MR de Discos poderia ser controlado por uma corrente elétrica, gerando com precisão um binário de travagem. Este sistema foi aplicado no modelo de $\frac{1}{4}$ de veículo a funcionar como *ABS* e o controlo foi realizado utilizando o *Sliding Mode Control*.

(Dariush Ghorbany, 2011), para além de explicar as diferenças entre os tipos de sistema de suspensão, aplicou-os ao contexto dos veículos e explicou os seus diferentes modos de funcionamento, mais concretamente o modo de Corte, o modo de Válvula e o modo de Aperto. Segundo (Dariush Ghorbany, 2011; Karakoc, 2007; Park et al., 2006; Poznić et al., 2012; Rossa et al., 2014), para que o fluído MR funcione como travão, este deve estar a funcionar no modo de corte, gerando um binário de travagem. (Dariush Ghorbany, 2011) na sua dissertação, implementou o modelo matemático de $\frac{1}{4}$ de veículo e simulou o seu sistema de suspensão e amortecimento, com base no sistema de suspensão semi-ativa, passiva e ativa, utilizando o modelo de Bingham, o modelo de *Bouc-Wen* e o modelo de Dahl.

(Poznić et al., 2012), implementou o travão MR utilizando o modelo de Bingham, onde os diversos tipos de travões MR foram considerados. Na investigação foram estudadas e comparadas as equações matemáticas dos binários e a complexidade do seu design para travões MR do tipo *Drum*, *Drum* invertido, múltiplos discos, um único disco e *T-shaped* rotor. O travão *T-shaped* rotor foi considerado o que possuía melhor performance e fornecia o maior binário de travagem. Mas, apesar destas vantagens, possuía muita inércia e maior complexidade na sua produção, fazendo com que o travão de um disco e múltiplos discos fossem considerados os melhores, pois possuem pouca inércia, funcionam num intervalo razoável de frequências e fornecem um bom binário de travagem.

Uma comparação semelhante foi realizada por (Rossa et al., 2014), que afirmou que a performance do travão MR varia com a sua geometria. Este utilizou o modelo de Bingham para os travões *Drum* e Disco e, modelou-o em função das suas características geométricas, assumindo que possuíam de facto uma relação linear. Implementou as equações matemáticas da densidade de binário, da eficiência, da controlabilidade e da largura de banda associadas à este sistema. Na comparação entre o travão *Drum* e o travão Disco, concluiu que a densidade de binário seria aproximadamente a mesma, que a eficiência do travão de Disco é claramente superior à do travão de *Drum* e, que o travão de Disco apresenta maior controlabilidade quando comparado com o travão de *Drum*.

Capítulo 3

Modelação, Controlo e Energia

3.1 Introdução

Neste capítulo, foi realizada uma descrição do aerogerador em estudo, a descrição teórica do sistema de medição do ângulo de *Yaw* do Aerogerador, do sistema de medição da velocidade e direção do vento (dados meteorológicos), do sistema de aquisição da potência elétrica produzida pelo aerogerador, do sistema de comunicação e sincronismo, do modelo de travão MR escolhido para ser controlado e do fluído MR . Os sistemas de aquisição de dados foram posteriormente acoplados ao Aerogerador, ao Anemómetro de Davis e aos condutores de produção elétrica, de forma a serem retirados dados reais para a criação do modelo real e do modelo ideal de potência elétrica. Para a análise desses dados em tempo real, foi implementado um sistema de monitorização com acesso a uma base de dados, na qual estes foram gravados e posteriormente organizados. Estes passos estão representados na Figura 3.1, que permite entender os passos seguidos e os objetivos a atingir.

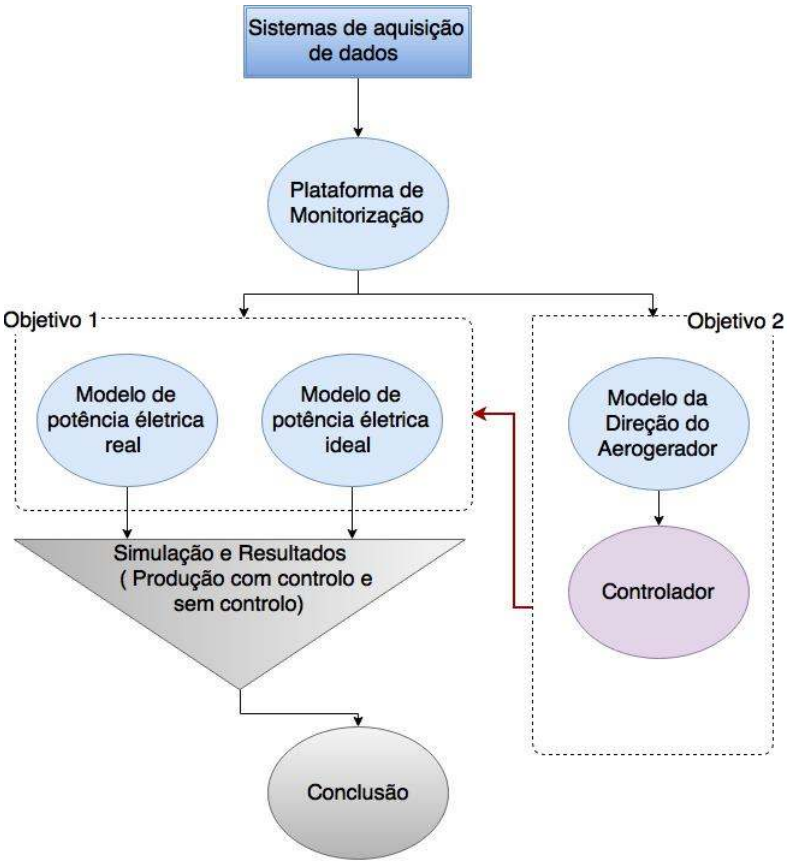


Figura 3.1: Arquitetura de alto nível da dissertação - Implementação.

3.2 Aerogerador de pequena potência

O aerogerador representado na Figura 3.2, o qual se pretende modelar, é constituído por um gerador de pequena potência da empresa *Yangzhou Shenzhou Wind-Driven Generator Co., Ltd*, modelo *FD3.6-2000-10L*, (Afonso, 2010), que está ligado a um rectificador/inversor que, por sua vez, encontra-se ligado à rede elétrica do edifício. As especificações do aerogerador eólico e a sua curva de potência estão representados na Tabela 3.1 e na Figura 3.3 consecutivamente. (Afonso, 2010; Standard, 2004)

Os dados da estação meteorológica associada a este sistema, eram recebidos através do *Datalogger Em50*, Figura 3.4, com um intervalo de medição de 60 segundos (segundo o fabricante), embora esse tempo possa ser reduzido até 10 segundos, caso seja configurado para tal, (Afonso, 2010). Estes dados são lidos em milivolt e convertidos para o formato *raw*, sendo posteriormente guardados.

Figura 3.2: Aerogerador localizado no Departamento de Engenharia Eletrotécnica , (Afonso, 2010).

Tabela 3.1: Especificações do Aerogerador

Parâmetro	**Valor**
Potência nominal [W]	2000
Tensão nominal [V]	120
Diâmetro das pás do rotor [m]	3.2
Velocidade do vento de arranque [m/s]	2
Velocidade do vento nominal [m/s]	9
Velocidade do vento de segurança [m/s]	16
Rotação nominal das pás [rpm]	400
Gerador	Alternador de Magnetos Permanentes
Material das pás	Fibra de vidro
Número de pás	3

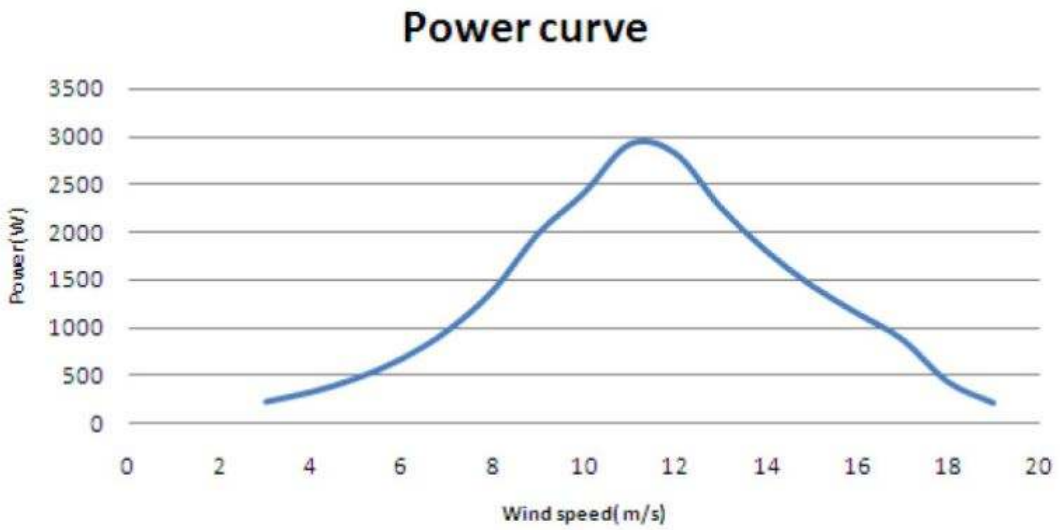


Figura 3.3: Potência gerada pelo aerogerador de pequena potência em função da velocidade do vento, (Afonso, 2010).

Figura 3.4: Datalogger Em50 - utilizado para aquisição dos dados da estação meteorológica, (Afonso, 2010).

3.3 Medição do Ângulo de Yaw do Aerogerador

Nesta secção é apresentado o sistema de medição da direção do aerogerador, tanto do ponto de vista de aquisição e transmissão de dados, como do sincronismo. Para a medição da direção do aerogerador foi necessário utilizar sensores que permitissem medir o ângulo de *Yaw*. Estes sensores podem conter um giroscópio, um acelerómetro e um magnetómetro. Os sensores que possuem magnetómetros são, tipicamente, utilizados quando se pretende ter como referência os polos magnéticos da terra, como é o caso de dispositivos que necessitam do cálculo correto do ângulo de *Yaw*. Foram testados e validados dois sensores em funcionamento com o Arduíno Uno, sendo que um deles possuía um magnetómetro (AltIMU-10 v4) e o outro não (MPU-6050). O esquema geral de funcionamento deste sistema é apresentada nas Figuras 3.5 e 3.6, considerando a aplicação dos dois diferentes sensores de medição.

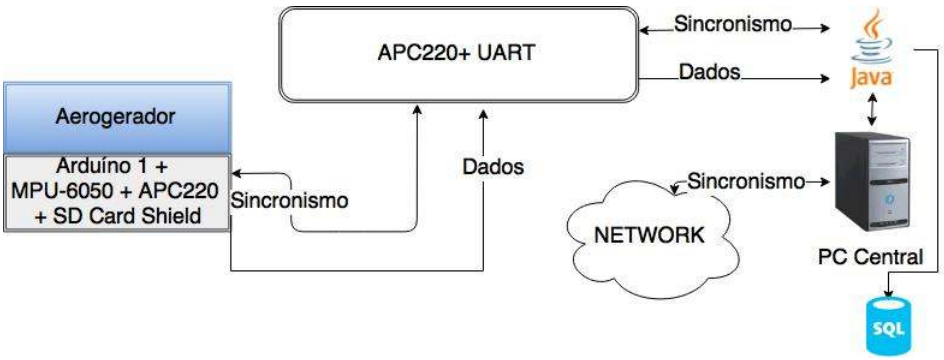


Figura 3.5: Esquema representativo da aquisição de dados do Aerogerador (com base no MPU-6050) e sincronismo proposto.

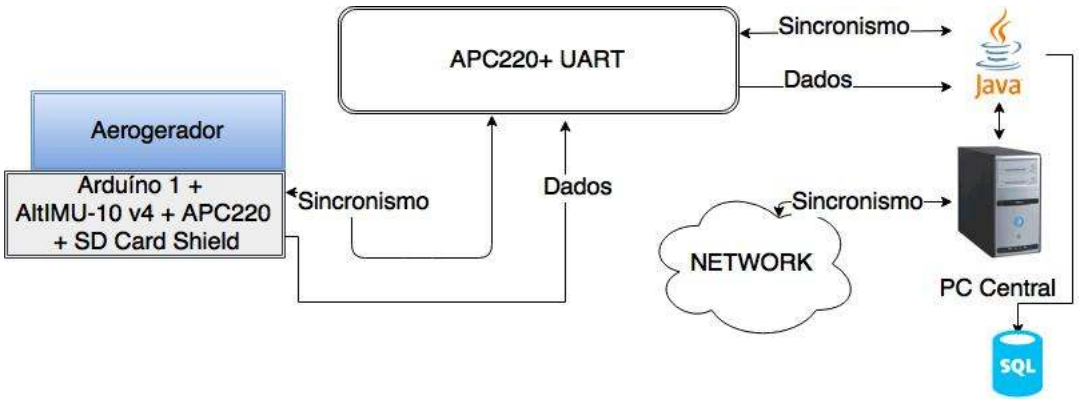


Figura 3.6: Esquema representativo da aquisição de dados do Aerogerador (com base no AltIMU-10 v4) e sincronismo proposto.

Quanto ao sensor MPU-6050, como referido no Capítulo 2, a unidade do giroscópio é [°/*sec*], com fatores de escala de ±250/500/1000/2000 [°/*sec*]. De forma a serem obtidos melhores resultados o fator de escala foi redefinido, utilizando o comando

```
accelgyro.setFullScaleGyroRange(MPU6050_ GYRO_ FS_ X)
```

onde $MPU6050\_GYRO\_FS\_X$ podia ser definido como $0x00$, $0x01$, $0x02$ e $0x03$ (Full-Scale Ranges: FS_ SEL=0, FS_ SEL=1 , FS_ SEL=2 e FS_ SEL=3), segundo a biblioteca do $MPU-6050$ disponibilizada por Jeff Rowberg, no ficheiro "$MPU6050.h$". Após a definição da sensibilidade e do fator de escala, o sensor foi calibrado e passou-se à fase de medições.

Os valores obtidos diretamente do sensor dependiam da sua sensibilidade e do seu modo de uso, sendo a conversão para $[graus(°)]$ realizada pelas Equações 3.1, 3.2 e 3.3, (Ave, 2013).

$$Gyro_X = \frac{Gyro_{X_{Medido}}}{16,4} \tag{3.1}$$

$$Gyro_Y = \frac{Gyro_{Y_{Medido}}}{16,4} \tag{3.2}$$

$$Gyro_Z = \frac{Gyro_{Z_{Medido}}}{16,4} \tag{3.3}$$

Os resultados obtidos com a utilização do MPU-6050 foram validados após a sua comparação com resultados do Encoder Incremental (Eltra EL-ER 58C) e com os resultados obtidos de um Tacómetro.

Aquando a utilização do sensor AltIMU-10 v4, foi utilizado o código disponibilizado em (Github, 2017), "minimu-9-ahrs-arduino", onde as sensibilidades do acelerómetro e do magnetómetro e, do giroscópio, estavam definidas como sendo $AFS\_SEL = 2$ (correspondente a $8g$) e $FS\_SEL = 3$ (correspondente a ± 2000), consecutivamente. O AltIMU-10 v4 foi validado de acordo com o método que será demonstrado no Capítulo 5 - Secção 4.1.

Os dados adquiridos foram enviados para o PC Central através de módulos de comunicação RF, que tinham também como propósito manter o sincronismo entre o Arduíno 1, o Arduíno 2 e o PC Central.

3.4 Medição da velocidade e direção do vento

No que toca a medição da velocidade e da direção do vento, foi utilizado o anemómetro de Davis que permitiu a aquisição de dados da velocidade do vento através da contagem dos pulsos recebidos. Quanto à direção do vento, esta foi obtida através de valores adquiridos de um potenciómetro associado ao cata-ventos do anemómetro de Davis que, dependendo da direção do vento, possuía um valor distinto de resistência elétrica e por isso um valor distinto de tensão elétrica. Este cata-ventos possui o valor de resistência de $20K\Omega$ quando está virado para norte (0/360°) e $10K\Omega$ quando está virado para sul, 180°. As especificações do anemómetro de Davis estão representadas nas Tabelas 3.2 e

3.3.

Tabela 3.2: Saídas do Anemómetro de Davis,(Davis Instruments, 2013).

Condutores	**Especificação**
Preto	Velocidade do vento
Vermelho	GND
Verde	Direção
Amarelo	Tensão de alimentação

Tabela 3.3: Escala, precisão, resolução e tempos de amostragem do Anemómetro de Davis, (Davis Instruments, 2013).

Anemómetro Davis	**Especificações**
Velocidade do vento	0,5 - 89 m/s
Direção do vento	0-360 º
Precisão - Velocidade do vento	0,1 m/s
Precisão - Direção do vento	7 º
Resolução - Velocidade do vento	0,1 m/s
Tempo de amostragem da Velocidade do vento	2,25 segundos
Tempo de amostragem da Direção do vento	1 segundo

É importante referir que o anemómetro de Davis fornece a velocidade do vento com um tempo de amostragem de 2, 25 segundos, enquanto que a direção do vento possui o tempo de amostragem de 1 segundo, (Davis Instruments, 2013).

Na Figura 3.7 é demonstrado o esquema representativo do sistema de medição da velocidade e da direção do vento. O Arduíno 2 foi responsável pela aquisição e tratamento dos dados da estação meteorológica. Este realizava a conversão dos pulsos recebidos do anemómetro em velocidade do vento, o cálculo da direção do vento, enviando posteriormente os dados para o PC Central.

A ligação do Anemómetro de Davis ao Arduíno 2 foi realizada através dos condutores do cabo *RJ-11*, que forneceram a informação relativa à direção, à velocidade, à alimentação elétrica e ao GND do Anemómetro, Figura 3.8.

Ainda na Figura 3.8, a porta digital 2 recebia os pulsos vindo do anemómetro e calculava a velocidade linear do vento em *Miles per hour* através da Equação 3.4,

$$V_{mph} = \frac{P_{Pulsos}}{T_{vel}} 2,25 \tag{3.4}$$

onde V_{mph} é a velocidade linear do vento, P_{Pulsos} é o número de pulsos contados e T_{vel} o período de amostragem em segundos, (Davis Instruments, 2013).

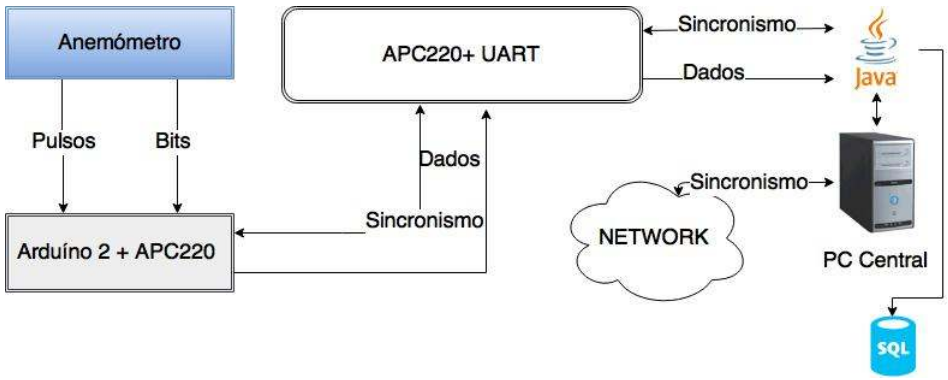


Figura 3.7: Esquema representativo da aquisição de dados da estação meteorológica e sincronismo.

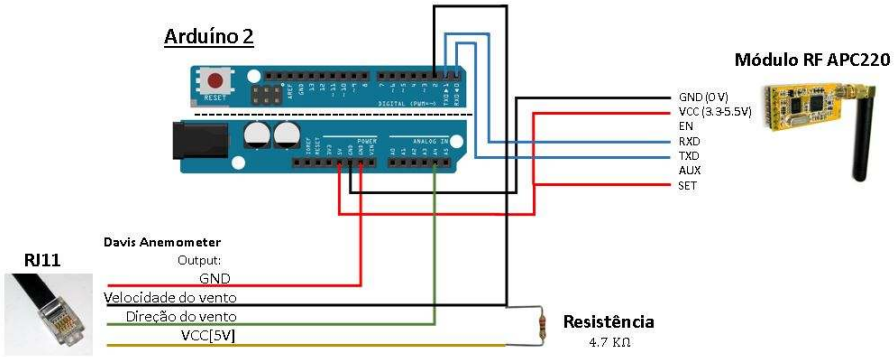


Figura 3.8: Esquema de ligação do Arduíno 2 na aquisição de dados da estação meteorológica.

Deve-se ressalvar, que o circuito de velocidade do vento é um interruptor que apenas é ativado quando os copos do Anemómetro completam uma volta de 360°. Do ponto de vista elétrico este é o GND quando realiza uma volta completa e $5V$ durante todo o resto do percurso. A resistência acoplada neste circuito faz com que o pin passe de $2V$ para $5V$ quando o interruptor está aberto, fazendo com que a tensão elétrica se mantenha fixa e evitando valores errados de velocidade do vento. A contagem do número de vezes que o pin representava o GND foi utilizado para calcular a velocidade do vento.

Quanto à direção do vento, esta foi lida pela porta analógica 4. Os valores recebidos por essa porta coincidem com números inteiros entre 1 e 1023, (Cactus.io, 2016), sendo que 1023 equivale a 360°, que por sua vez coincide com o potenciómetro a $20K\Omega$. Seguindo este raciocínio, 180° equivale a 511,5, que por sua vez, coincide com o potenciómetro a $10K\Omega$, Figura 3.9.

A conversão dos valores recebidos pela porta analógica 4 para $[graus(°)]$ é dada pela Equação 3.5.

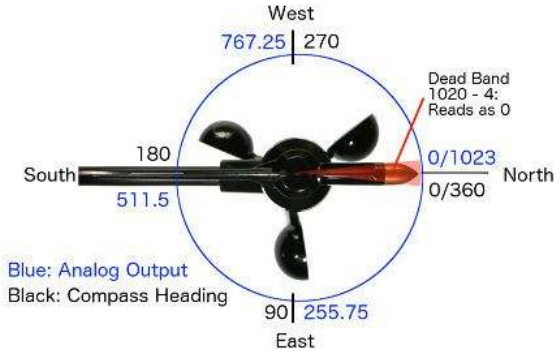


Figura 3.9: Relação entre os números inteiros recebidos e a posição angular do vento, (Cactus.io, 2016).

$$Wind_{Direction} = AnalogValue * \frac{1023}{360} \tag{3.5}$$

em que $AnalogValue$ corresponde ao valor recebido na porta analógica do Arduíno Uno e $Wind_{Direction}$ é o valor da direção do vento em graus.

Após a conversão dos dados para as unidades desejadas (m/s e $graus(°)$), foram enviados para o PC Central através de módulos de comunicação RF. Os referidos tinham também como objetivo manter o sincronismo dos dados adquiridos com o Arduíno 1 e o PC Central, à semelhança do que acontecia na Secção 3.3.

3.5 Sistema de Medição da Potência Elétrica Produzida

O Analisador de Qualidade de Energia *Fluke 43B*, foi utilizado na medição da potência elétrica produzida, Figura 3.10.

Figura 3.10: Analisador de Qualidade de Energia Fluke 43B.

Este permite medir a corrente e a tensão elétrica, as potências ativas, passivas e aparentes, o fator de potência e a frequência das ondas e, enviar alguns destes dados para o PC2 através de uma porta de comunicação *Serial RS*232. Porém, este modelo específico da

Fluke implica a utilização de drivers apropriados para a leitura deste aparelho, aquando a utilização do Matlab para a sua leitura, (MathWorks, 2017a).

Existia também, a possibilidade de se utilizar o módulo MAX232 ou o Shield RS232/485 para o Arduíno Uno, no entanto existia uma terceira solução a um custo reduzido. Esta terceira solução era a utilização do Software designado para o modelo do Fluke 43B, *FlukeView* 3.34, que permite adquirir os dados do Analisador de Qualidade de Energia e armazena-los num ficheiro no PC2, Figura 3.11. Neste caso era essencial que o ficheiro estivesse no formato de texto, Figura 3.12, de forma a se poder realizar uma leitura rápida.

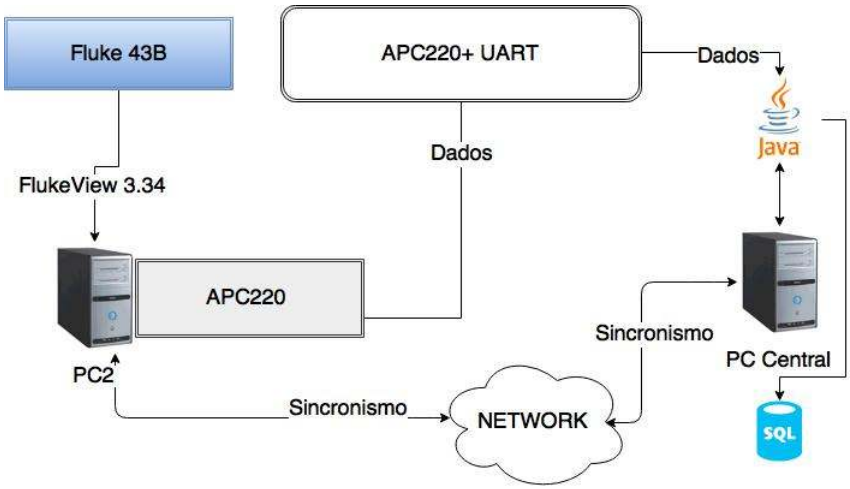


Figura 3.11: Esquema representativo da aquisição de dados da produção elétrica e sincronismo.

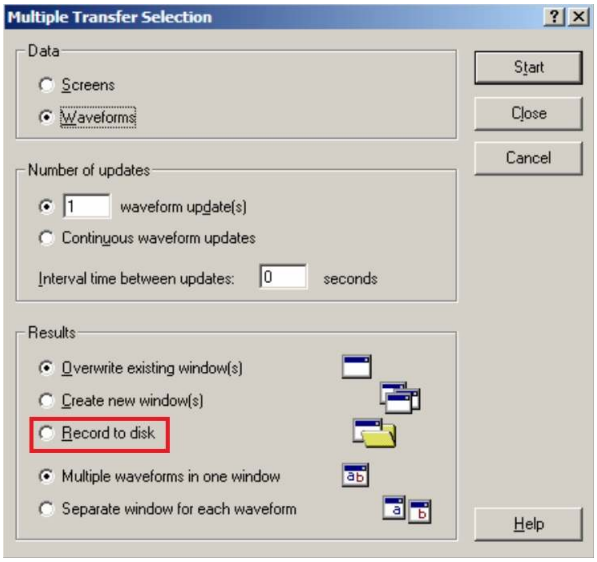


Figura 3.12: Software FlukeView 3.34, opção de armazenamento dos dados do Fluke 43B num ficheiro no PC2.

Previamente ao armazenamento dos dados do Fluke 43B no PC2, foi necessário definir de que forma os dados seriam lidos, ou seja, se seriam lidos continuamente ou se existiria um número pré-definido de dados. Posteriormente seria necessário definir o tempo de amostragem. Por exemplo, caso se optasse pela leitura continua e um tempo de amostragem de dados de 20 segundos, este criaria um ficheiro "rec1.txt" com os dados adquiridos. Após 20 segundos iniciaria uma nova aquisição de dados para um segundo ficheiro, "rec2.txt", e assim consecutivamente.

Este processo pode ser representado pelo esquema da Figura 3.13, onde se observa que é possível adquirir dados de forma continua ao longo de um período de tempo, até ser dada a ordem de paragem ou, na situação de terminar o espaço no PC2. É importante referir que a potência elétrica adquirida foi o seu valor instantâneo, uma vez que tanto a corrente elétrica como a tensão elétrica eram dados instantâneos.
Estes dados seriam posteriormente enviados para o PC Central por intermédio de um módulo de comunicação RF. Quanto ao sincronismo com os restantes sistemas de aquisição de dados, este foi realizada pelo facto do PC2 se encontrar na mesma rede Internet que o PC Central.

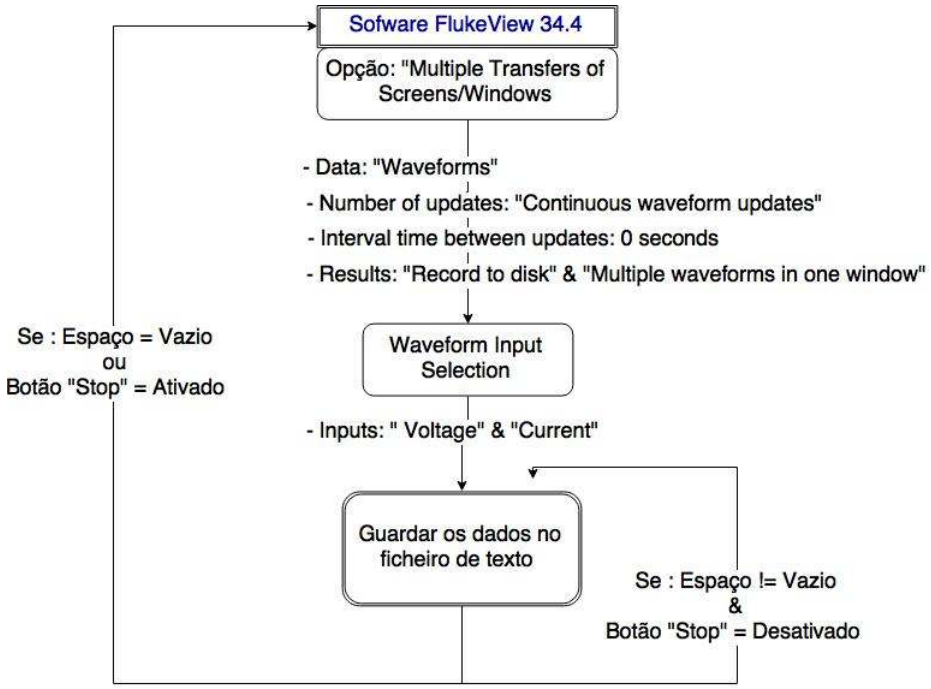


Figura 3.13: Esquema de aquisição de dados do Fluke 43B e armazenamento no PC2.

3.6 Comunicação e Sincronismo

O sincronismo entre os sistemas de aquisição de dados e o PC Central era necessário, de modo a manter coerência nos dados recebidos. Por exemplo: no instante t o aerogerador estaria na posição angular θ, com uma velocidade angular ω, a produzir a potência P e o vento possuiria uma velocidade v na direção D_{vento}.

Caso os Arduínos e o PC2 não estivessem síncronos, não seria possível relacionar os diferentes dados obtidos e por este motivo, o sistema de sincronismo wireless era necessário. Nesta situação foram testados dois sistemas de sincronismo, utilizando dois módulos RF, Figuras 3.14 e 3.15.

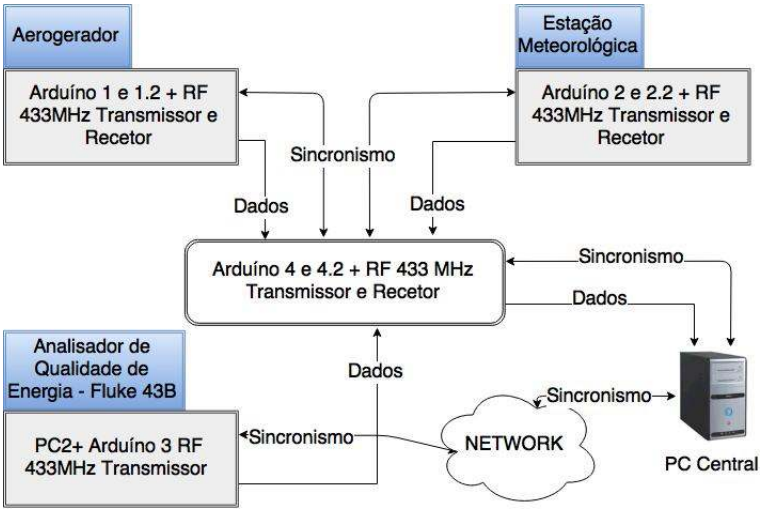


Figura 3.14: Esquema do sistema de aquisição de dados e sincronismo testado, módulo RF 433MHz.

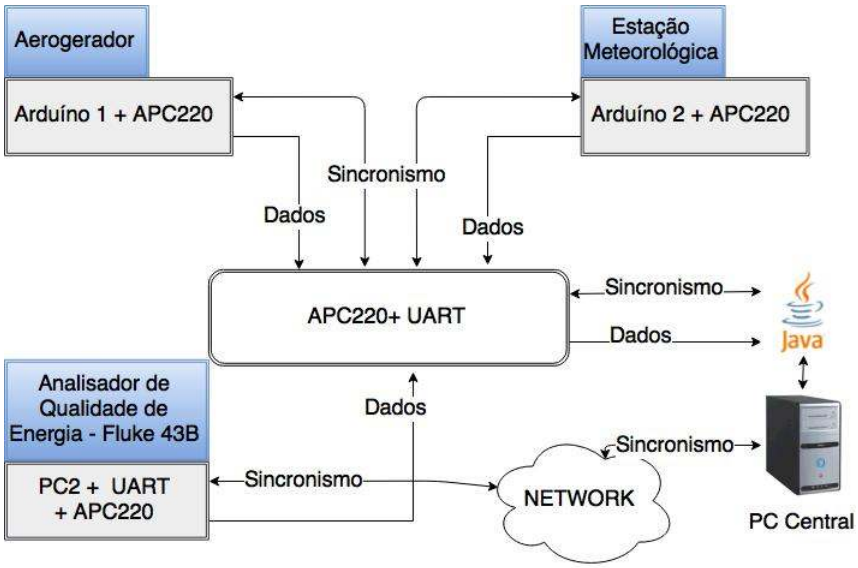


Figura 3.15: Esquema do sistema de aquisição de dados e sincronismo, módulo APC220.

Com a utilização do Módulo RF 433MHz (Capítulo 2 - Secção 2.14.1) para comunicação e sincronismo, os Arduínos 1, 2 e 3 deveriam conectar via wireless com o Arduíno 4, de modo a estarem síncronos entre si e com a rede Internet (Figura 3.14). Após a ligação dos módulos aos seus respetivos Arduínos, foi necessário programa-los para que enviassem e/ou recebessem os dados enviados pelo transmissor. Para efeitos de teste, o Arduíno 1 enviou dados para o Arduíno 4, de acordo com o esquema da Figura 3.16.
Esta programação foi realizada tendo por base o código disponibilizado pelo "FILIPEFLOP ", cujo programa tinha como objetivo a troca de dados entre os módulos, utilizando a biblioteca *VirtualWire*.

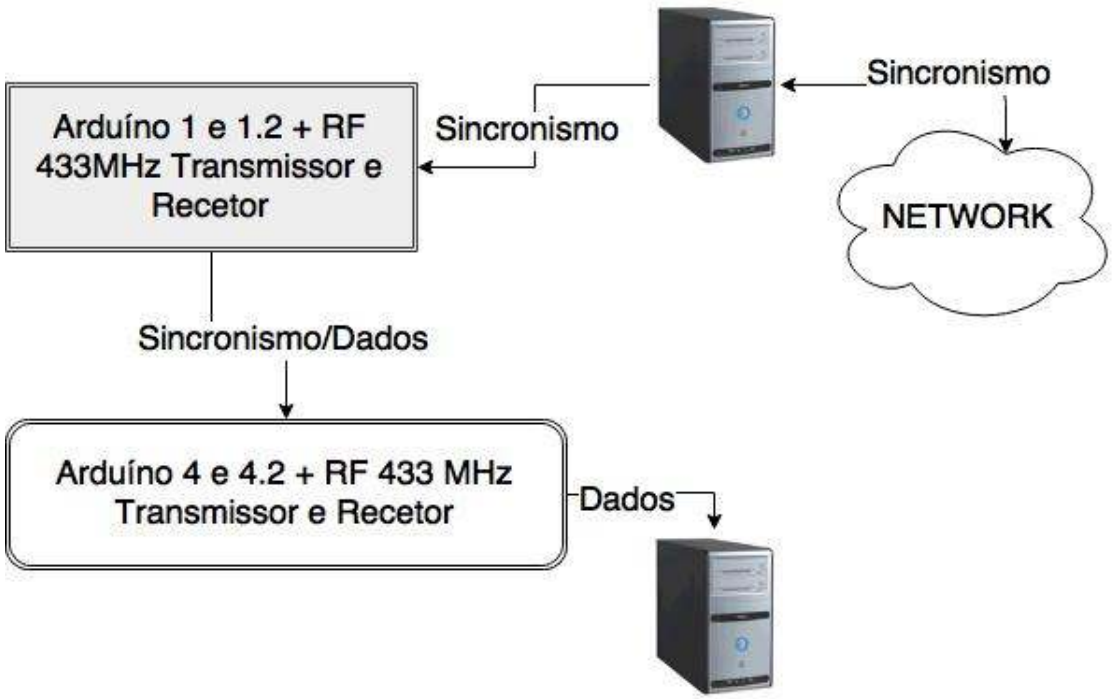


Figura 3.16: Esquema de transmissão e receção de dados utilizado para efeitos de teste de comunicação.

Relativamente à utilização do Módulo APC220 (Capítulo 2 - Secção 2.14.2), foi testado o esquema completo da Figura 3.15, devido à simplicidade no envio e receção de dados (Comparativamente ao módulo RF 433Mhz). Previamente à montagem do referido esquema, foi necessário configura-los de forma a que todos possuíssem as mesmas características e se pudessem comunicar. Para interação com este módulo de comunicação foi criada uma aplicação em JAVA no PC Central, de modo a enviar os dados de sincronismo e receber os dados dos respetivos sistemas.

3.7 Modelação do Aerogerador - Modelo Real e Modelo Ideal de Potência Elétrica

Na criação de um modelo que estimasse a potência elétrica gerada pelo aerogerador, foram utilizadas as redes neuronais. Com base nas diversas entradas e saídas obtidas, este criaria um sistema em caixa-preta que seria capaz de estimar a saída desejada, dependendo dos parâmetros de entrada, Figura 3.17.

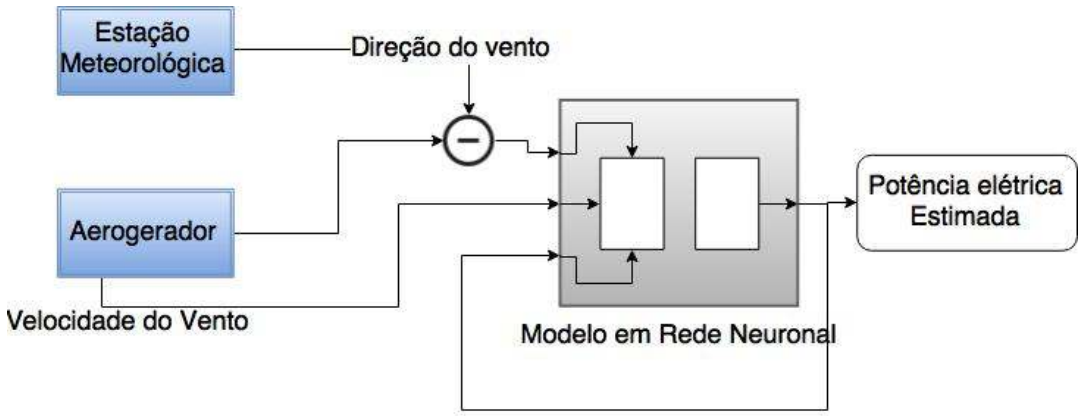


Figura 3.17: Representação do modelo real.

As redes neuronais foram utilizadas, porque o sistema possuía um comportamento não linear e não eram conhecidos alguns aspetos da sua dinâmica (binário produzido por uma determinada força do vento, vibrações e inclinações que pudessem influenciar a orientação do aerogerador, entre outros).

A utilização deste método implica que se possuam amostras relativas ao funcionamento do sistema que se deseja estimar, tendo sido utilizadas as amostras recolhidas pelos sistemas de aquisição de dados. Quanto ao número de neurónios da camada interna e externa, estes foram definidos de acordo com o número de entradas e saídas que se pretendiam para o modelo.

No que diz respeito ao modelo ideal, este baseou-se em equações matemáticas que descreviam a potência do vento, a potência mecânica aproveitada do vento pelo aerogerador e a potência elétrica gerada. Estas potências possuem uma relação entre elas descrita por rendimentos, sendo estes indicadores da quantidade da potência anterior que foi possível aproveitar, Figura 3.18.

Sabendo que a potência fornecida pelo vento P_{vento} pode ser dada pela Equação 3.6, caso fosse conhecida a potência mecânica P_{mec} seria possível calcular o C_p, com base na Equação 3.7. No entanto, sendo este um modelo teórico, o C_p deveria ser também teórico.

$$P_{vento} = \frac{1}{2} A \rho v^3 \tag{3.6}$$

$$C_p = \frac{P_{mec}}{P_{vento}} \tag{3.7}$$

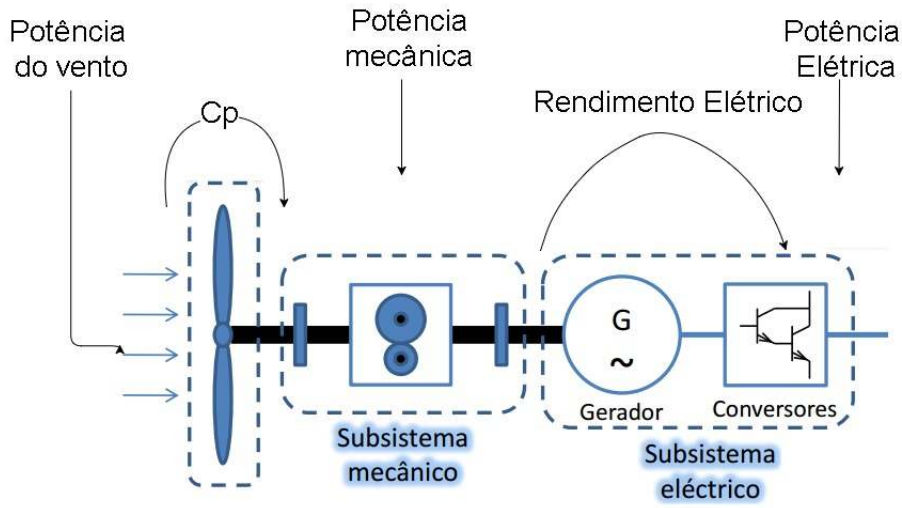


Figura 3.18: Rendimentos associados às potências extraídas do vento.

Trabalhando no mesmo sistema, (Afonso, 2010), utilizou um rendimento elétrico η (gerador síncrono e conversor eletrónico de potência) de 90% e concluiu que o C_p máximo para este aerogerador era aproximadamente 40%. Considerando uma redução no rendimento causado essencialmente pelo tempo, foi admitido um rendimento elétrico ligeiramente inferior. No entanto para o C_p foi assumido o seu valor máximo, uma vez que este depende de fatores associados à aerodinâmica das pás (ângulo de *Pitch*, coeficientes de sustentação e arrastamento, entre outros) e à posição do aerogerador face ao vento. Para a fase da estimação da potência ideal produzida, deveria ser calculada a densidade da massa de ar ρ, que é dada pela Equação 3.8 e que depende das Equações 3.9 e 3.10.

$$\rho(P_r, Temp) = \frac{353 P_r}{Temp} \tag{3.8}$$

Pr é a pressão do ar em atm e $Temp$ é a temperatura do ar em K.

$$P_r(H_{TP}) = \exp(-1,185 \times 10^{-4} H_{TP}) \tag{3.9}$$

H_{TP} é a altitude em metros.

$$\Delta Temp(H_{TP}) \approx -9,8°C/Km \tag{3.10}$$

Tendo sido calculado ρ e P_{vento} e, conhecendo o η e o C_p foi possível estimar a potência elétrica ideal com a Equação 3.11.

$$P_{elec} = P_{vento} C_p \eta = P_{mec} \eta = \frac{1}{2} C_p \eta A \rho v^3 \tag{3.11}$$

Uma vez que é conhecido o comportamento interno deste modelo, este pode ser considerado um modelo em caixa-branca.

3.8 Travão Magnetoreológico – Modelo Matemático

O modelo real e o modelo ideal de potência elétrica diferem entre si essencialmente devido ao C_p, que foi assumido como constante e máximo no modelo ideal. Como referido anteriormente, uma das implicâncias do C_p máximo é o erro de *Yaw* ser nulo (aumentando a produção elétrica), o que não se verifica nas condições reais. Por este motivo foi necessário implementar um sistema de travagem, de forma a estabilizar o aerogerador quando este se aproximasse da posição ideal (direção do vento). Este sistema deveria ser controlado de modo a atuar quando o erro de *Yaw* fosse superior a um erro máximo aceitável.

Nesta secção foi introduzido um travão magnetoreológico e foram estudados alguns dos seus modelos matemáticos. Estes modelos paramétricos permitem que com base em valores numéricos, que representam condições reais, seja possível simular e observar um comportamento idêntico ao observado na prática, (Oliveira, 2015; Paschoal, 2011).

3.8.1 Modelo de Bouc-Wen

Segundo (Dariush Ghorbany, 2011; Oliveira, 2015; Paschoal, 2011), o modelo de Bouc-Wen tem sido utilizado quando se pretendem modelar sistemas com histerese, ou seja, modelar casos em que o modelo de Bingham, que foi o primeiro a descrever o comportamento reológico dos materiais, não seja o mais adequado.

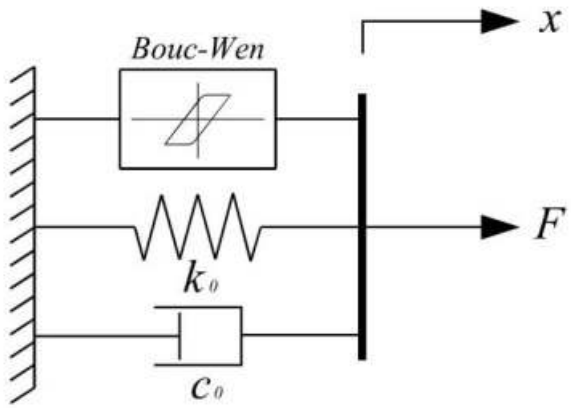


Figura 3.19: Modelo de Bouc-Wen, (Oliveira, 2015).

Na Figura 3.19 está representado o esquema do modelo, onde é possível observar os coeficientes que influenciam a força total produzida por este sistema, Equação 3.12,

$$F = c_0\dot{x} + k_0 x + \alpha z \tag{3.12}$$

sendo F a força aplicada, K_0 a rigidez elástica, x o deslocamento, z a variável que representa o comportamento da histerese obtida a partir da Equação 3.13, α, β, γ, n e A os

parâmetros característicos do modelo que servem para ajustar a linearidade e a suavidade na inversão do sentido da velocidade.

$$\dot{z} = -\gamma \mid \dot{x} \parallel z \mid^{n-1} z - \beta \dot{x} \mid z \mid^{n} + A\dot{x} \tag{3.13}$$

Tendo em conta que se pretende travar um objeto com movimento rotacional, a força deve ser representada em função do binário, Equação 3.14,

$$T_{MR} = \alpha(I)z + C(i)\dot{\theta} \tag{3.14}$$

sendo $C(i)$ e $\alpha(I)$ descritos pelas Equações 3.15 e 3.16,

$$C(i) = C(i) = C_0 + C_1(I) \tag{3.15}$$

$$\alpha(I) = \alpha_0 + \alpha_1(I) \tag{3.16}$$

onde T_{MR} é o binário provocado pelo fluído MR, $C(I)$ é o coeficiente de amortecimento que depende da corrente elétrica aplicada I, θ o deslocamento ângular, z é a variável que descreve a histerese, agora descrita pela Equação 3.17, (Dariush Ghorbany, 2011).

$$\dot{z} = -\gamma \mid \dot{\theta} \parallel z \mid^{n-1} z - \beta \dot{\theta} \mid z \mid^{n} + A\dot{\theta} \tag{3.17}$$

3.8.2 Modelo de Bingham

Segundo (Paschoal, 2011), o modelo de Bingham (Figura 3.20), como referido anteriormente, foi o primeiro modelo a descrever o comportamento reológico dos materiais, com base na tensão de escoamento. Trata-se de um modelo visco-elástico que tem associada uma determinada variação de viscosidade e, por sua vez, uma variação de força e binário, de acordo com a intensidade do campo magnético a que está sujeito.

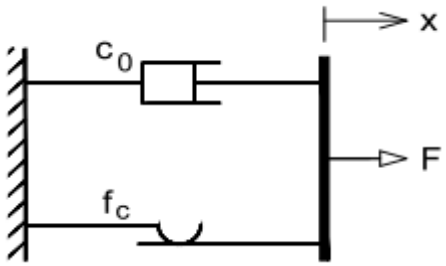


Figura 3.20: Modelo de Bingham , (Oliveira, 2015).

$$\tau = \tau_y(H) + \nu\dot{\gamma} \tag{3.18}$$

τ é a tensão de corte *Shear Stress*, H é o campo magnético, ν é a viscosidade que é calculada como sendo o declive da tensão de corte em função da razão de deformação (*Shear Rate*), $\dot{\gamma}$ a razão de deformação, τ_y tensão de escoamento, (Oliveira, 2015; Poznić et al., 2012).

A tensão de escoamento é responsável por causar uma determinada força descrita pela Equação 3.19,

$$F = c_0\dot{x} + f_c(H)sgn(\dot{x})) \tag{3.19}$$

sendo F a força aplicada pelo sistema, c_0 o coeficiente de amortecimento, $f_c(H)$ a força de atrito de Coulomb e sgn a função *Signal* representada pela Equação 3.20.

$$sgn(\dot{x}) = \begin{cases} -\dot{x} & \text{se } \dot{x} > 0 \\ +\dot{x} & \text{se } \dot{x} < 0 \end{cases} \tag{3.20}$$

Dada a sua simplicidade e ao facto do aerogerador não possuir dinâmicas rápidas sobre o eixo do *Yaw*, foi este o modelo escolhido para implementação. Porém, sendo este aplicado à rotação do aerogerador no eixo do *Yaw*, a força aplicada deve ser representada em função do binário, ou seja, o binário de travagem gerado a partir do campo magnético aplicado. Para tal, foi necessário analisar diversas abordagens, partindo da equação que representa a tensão de corte do modelo de Bingham, Equação 3.18, (Dariush Ghorbany, 2011).

3.9 Binário de Travagem

Diversos trabalhos foram realizados abordando de forma distinta o modelo de Bingham para travões de disco. Partindo do trabalho realizado por (Park et al., 2006), o binário fornecido por este modelo pode ser dado pela Equação 3.21,

$$T_{MR} = T_H + T_\nu \tag{3.21}$$

em que T_H e T_ν podem ser descritos pelas Equações 3.22 e 3.23 e, I em função do H pode ser obtido ao considerar a Equação 3.24,

$$T_H = \frac{2\pi}{3} N k \alpha (R_o^3 - R_i^3) I \tag{3.22}$$

$$T_\nu = \frac{\pi}{2h} N \nu (R_o^4 - R_i^4) \dot{\theta} \tag{3.23}$$

$$H = \alpha I \tag{3.24}$$

onde T_H é o binário que depende da tensão de escoamento, T_ν é o binário fornecido pela viscosidade e fricção do fluído, I é a corrente elétrica aplicada, R_i é o raio interno do disco, R_o é o raio externo do disco, N é o número de superfícies do disco em contacto com o fluído, α é o ganho proporcional , ν é a viscosidade e $\dot{\theta}$ é a velocidade angular do disco.

Uma outra abordagem é representada na Equação 3.25, onde o binário de travagem é calculado em função da tensão de corte, Equação 3.18, a velocidade angular é dada por ω, o binário causado pela força de fricção é T_{fric}, N é o número de superfícies em contacto com o fluído, ν é a viscosidade do fluído, R_o é o raio exterior do disco e g o comprimento do espaço preenchido pelo fluído, Figura 3.21.

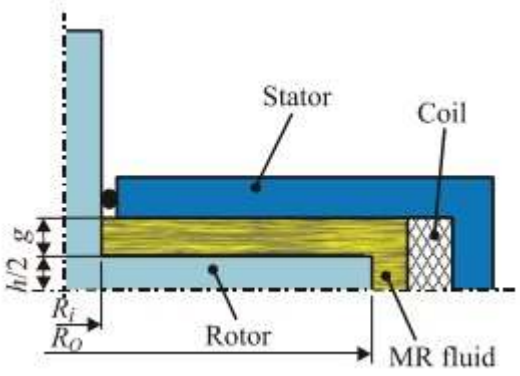


Figura 3.21: Representação simplificada de um travão MR de Discos, (Poznić et al., 2012).

$$T_{MR} = \pi N(\frac{4}{3}\tau R_o^3 + \nu\frac{\omega}{g}R_0^4) + T_{fric} \tag{3.25}$$

Esta abordagem foi implementada devido à simplicidade da Equação 3.25, por depender em grande parte de componentes físicas que podem ser representadas por constantes, (Poznić et al., 2012), assim como, devido ao facto da tensão de escoamento ser representada por um polinómio de terceira ordem, Equação 3.26,

$$\tau_y = K_1 B + K_2 B^2 + K_3 B^3 \tag{3.26}$$

em que B é o campo magnético induzido e K_i os coeficientes de regressão.

3.10 Fluído Magnetoreológico MRF-140CG

Foi escolhido um fluído MR monopolizado pela empresa *LORD Corporation*, *MRF-140CG Magneto-Rheological Fluid* (Figura 3.22), devido à sua elevada viscosidade comparativamente ao *MRF-132DG* e o *MRF-122EG*. Utilizando o *Datasheet* que descrevia a tensão de escoamento em função do campo magnético (Figura 3.23), o campo magnético induzido em função do campo magnético (Figura 3.24) e a tensão de corte em função da Razão de deformação (Figura 3.25), foi possível simular o comportamento do fluído.

Figura 3.22: MRF-140CG, (*LORD Corporation - MRF*).

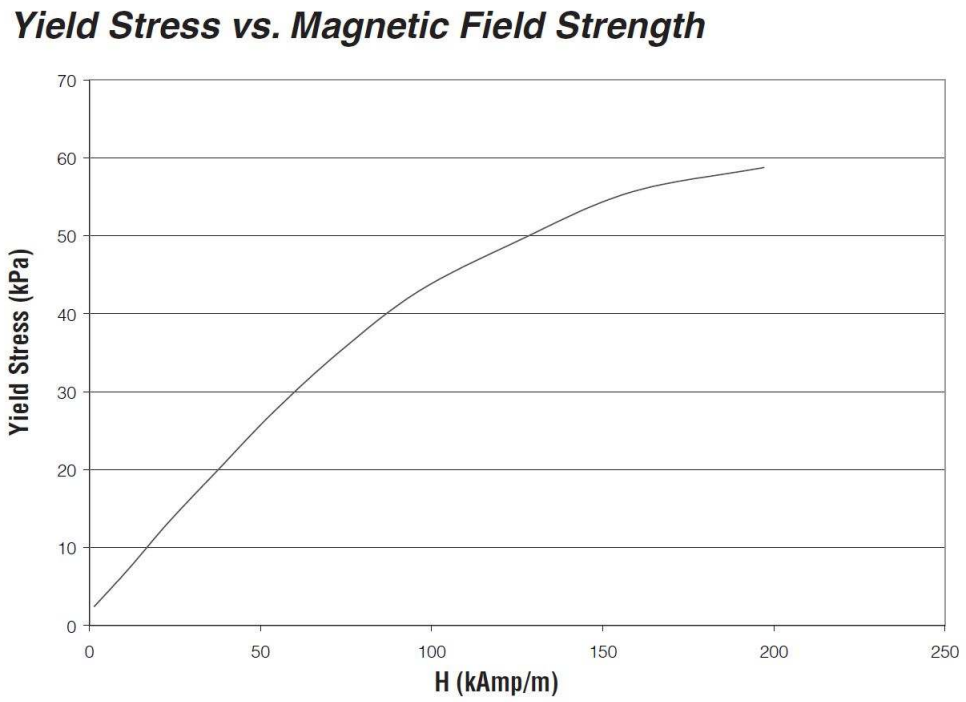


Figura 3.23: Tensão de escoamento em função do campo magnético, (Data, 2008).

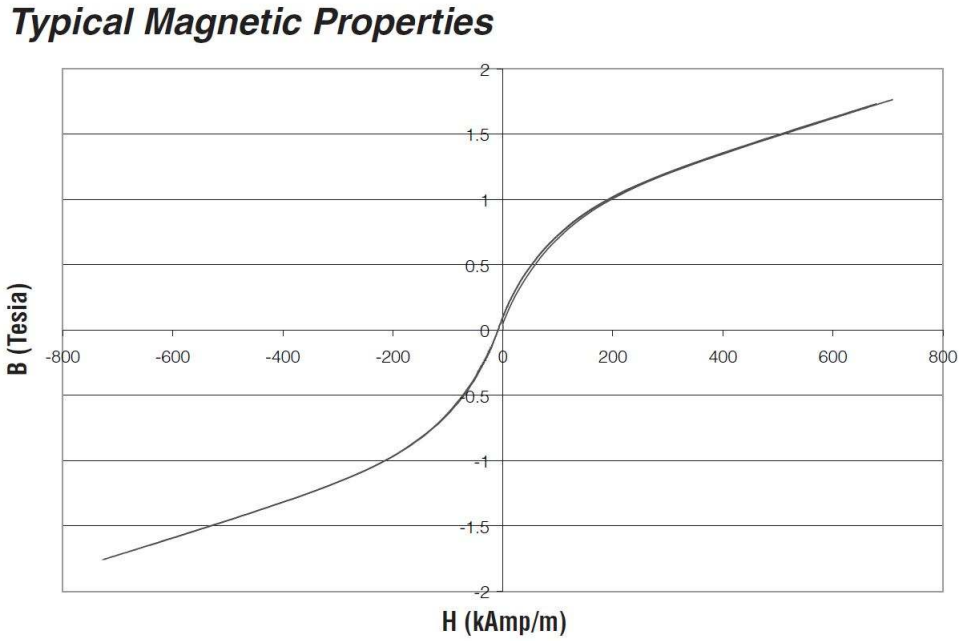


Figura 3.24: Campo magnético induzido em função do campo magnético, (Data, 2008).

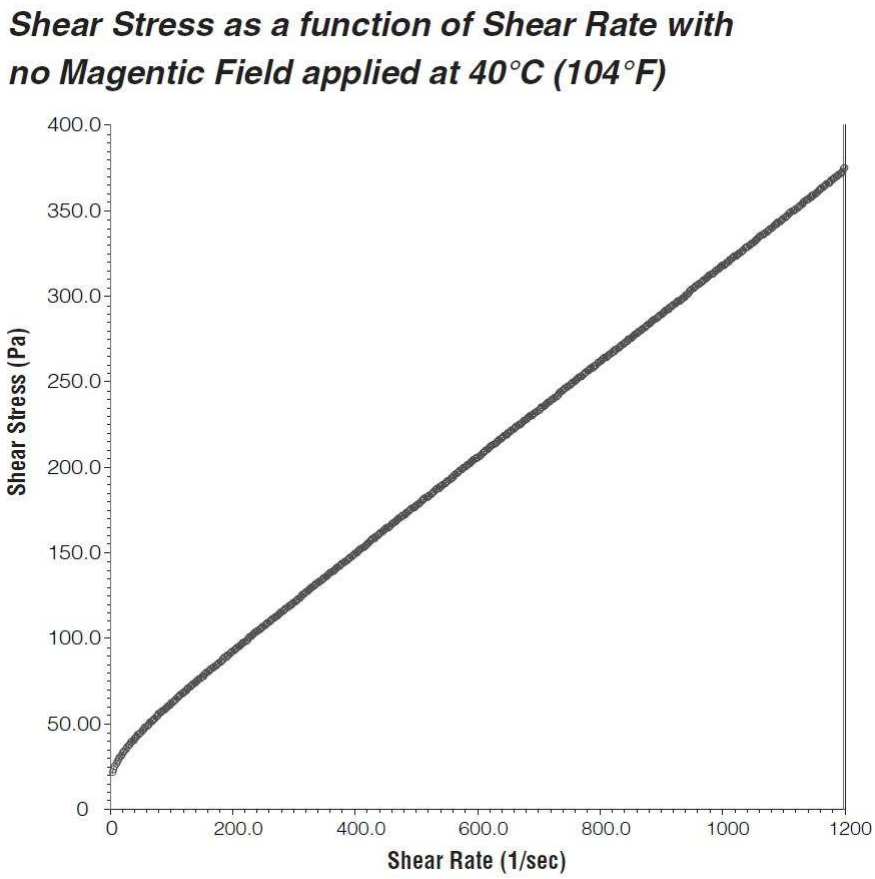


Figura 3.25: Tensão de corte em função da razão de deformação, (Data, 2008).

3.11 Controlador de Binário e de Posição

A criação de um controlador semi-ativo foi necessária de forma a manter o aerogerador na direção do vento. O intuito era travar o aerogerador quando este estivesse alinhado com a direção do vento e, por este motivo, foi utilizado o modelo de Bingham (Secção 3.8.2). Para a aplicação do controlador assumiu-se que parte da arquitetura do aerogerador de pequena potência foi modificada, de acordo com o representado na Figura 3.26. Note-se que o aerogerador em questão é um *Free-yaw drive*, tendo a arquitetura representada e explicada no Capítulo 2- Secção 3.2, no entanto este modelo de aerogerador (representado na Figura 3.26) serve para explicação dos objetivos, uma vez que a mudança de arquitetura proposta enquadra-se na torre que sustenta o aerogerador.

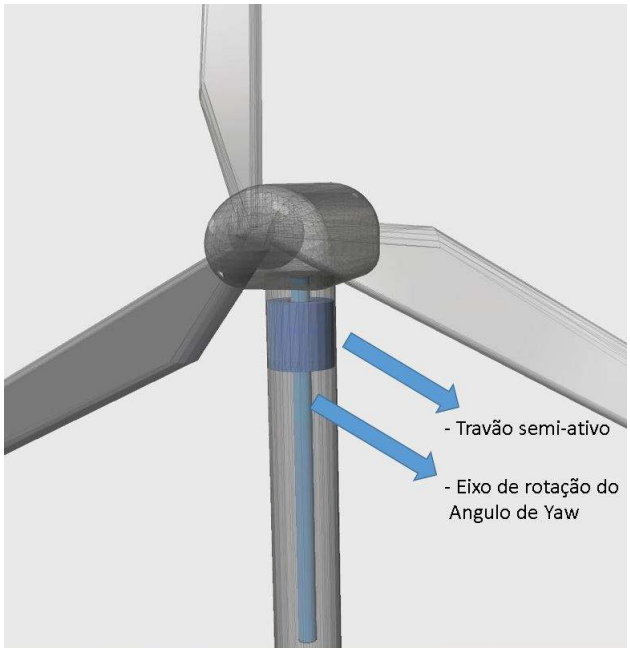


Figura 3.26: Proposta de modificação na estrutura física do aerogerador de pequena potência, adaptação do modelo de uma turbina eólica, (Chiodi, 2016), para a plataforma *AUTODESK 123D* .

Tendo sido instalado o travão MR na torre do aerogerador, foram implementados dois controladores difusos: o controlador difuso de binário e o controlador difuso de posição. O controlo difuso foi aqui aplicado devido a simplicidade na sua lógica de funcionamento.

O controlador de binário foi implementado de forma a que, com base num erro de binário (diferença entre o binário de referência e o binário atual do aerogerador), este gerasse uma corrente elétrica que alterasse o comportamento do fluído MR, dando origem a uma tensão de corte. Esta tensão de corte foi a responsável por criar um binário MR (Equação 3.25), de forma a minimizar o erro de binário, mantendo o binário total do sistema num valor de referência.

O binário de referência referido, foi criado pelo controlador de posição. Este controlador, dependendo do erro de posição, daria origem ao binário de referência, que iria

influenciar o travão MR tal como descrito anteriormente. Relativamente ao erro de posição, este representa a diferença entre a posição do vento (referência) e a posição atual do aerogerador, que surgiu como resposta do novo modelo criado, ou seja, o modelo em redes neuronais de movimento de rotação do aerogerador.

Tendo sido introduzido o sistema MR na torre do aerogerador, tornou-se possível calcular o binário de fricção T_{fric} (representado na Equação 3.25), uma vez que este está associado à rotação do aerogerador segundo o eixo de *Yaw* (Figura 3.27). Sendo o binário de fricção descrito pela Equação 3.27, (Leitão, 2012), este corresponde ao binário natural do eixo do aerogerador, dependente da massa em rotação e da aceleração angular (Figura 3.27). Quanto ao momento de inércia J, este foi calculado com base na Equação 3.28 visto o eixo de rotação em questão ser cilíndrico, o que permite a utilização da referida equação.

$$T_{fric} = J\frac{\partial^2 Yaw}{\partial t^2} \tag{3.27}$$

$$J = \frac{Massa \times R^2}{2} \tag{3.28}$$

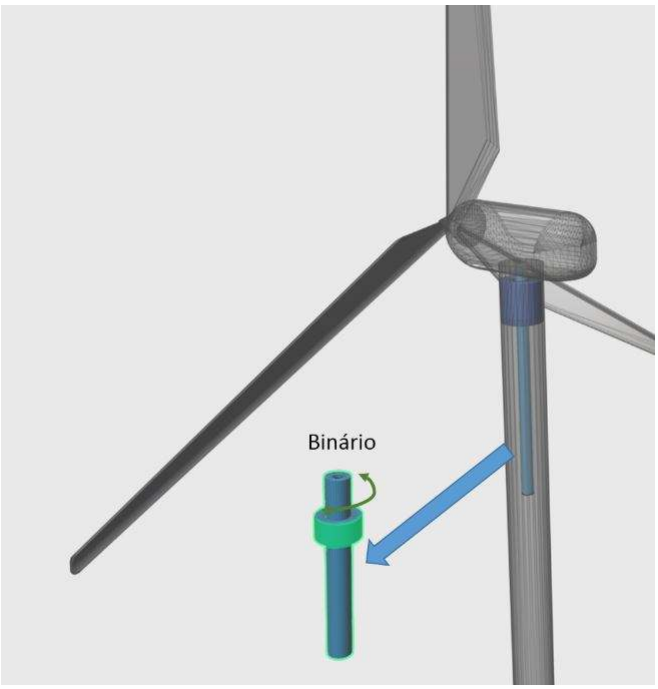


Figura 3.27: Binário associado ao eixo de rotação do aerogerador.

Capítulo 4

Aquisição e Monitorização de dados

Neste capítulo foram implementados os sistemas de aquisição de dados e o sistema de sincronismo, com o objetivo de serem obtidas amostras para a criação dos modelos de potência elétrica real e ideal. Uma plataforma de monitorização foi criada com o intuito de ser possível correlacionar e analisar as amostras obtidas.

4.1 Implementação do Sistema de Medição do Ângulo de Yaw do Aerogerador

Nesta fase de implementação foi utilizado o sensor MPU-6050 com o objetivo de obter a direção do aerogerador. Para tal, foi necessário programar o Arduíno 1 de forma a que este, para além de ler os valores fornecidos pelo sensor, também os armazenasse num cartão micro-SD introduzido num Shield para o Arduíno Uno. Estes dados deveriam posteriormente ser enviados para o PC Central, por intermédio de um módulo de comunicação RF (Figura 3.5). Este módulo de comunicação serviria para a transmissão dos dados adquiridos e para a receção de pacotes de sincronismo, de forma a manter uma coerência temporal entre o PC Central e o Arduíno 1. Deste modo, quando os dados chegassem ao PC central, este saberia a que instante de tempo corresponderia aquela informação. Quanto ao sensor MPU-6050, a escala de medição foi definida como sendo $FS\_SEL = 3$ (correspondente a 2000 [°/*sec*] e um fator de escala de 16,4), de acordo com o explicado na Secção 3.3 e, de seguida, passou-se ao processo de calibração do sensor. Foi compilado o código de calibração, (Ródenas, 2016), permitindo obter os seguintes valores de *Offset*:

- $Accel_x = 285$
- $Accel_y = -344$;
- $Accel_z = 1146$;

- $Gyro_x = 72$;
- $Gyro_y = 42$;
- $Gyro_z = 26$.

Estes *Offset* foram definidos no sensor com as seguintes linhas de comando:

```
accelgyro.setXAccelOffset(285);
accelgyro.setYAccelOffset(-344);
accelgyro.setZAccelOffset(1146);

accelgyro.setXGyroOffset(72);
accelgyro.setYGyroOffset(42);
accelgyro.setZGyroOffset(26);
```

Uma vez que o giroscópio fornecia valores em função de ±2000[°/*sec*] era necessário converte-los para [°/*sec*], utilizando para tal a Equação 3.3, uma vez que descreve a velocidade segundo o eixo do *Yaw*.

De forma a validar os resultados obtidos pelo sensor MPU-6050, foi utilizado um Encoder Incremental da *Eltra* , *EL58C300S5/28P10X6PR*, também conectado ao Arduíno 1, com o intuito de obter valores de velocidade angular de um motor DC de frequência variável. Para tal foi necessário programar o Arduíno, de forma a que a contagem dos pulsos do Encoder Incremental fosse realizada segundo a codificação X2, Figura 4.1, uma vez que esta fornecia os resultados com precisão e indicava o sentido de rotação. O sensor foi afixado ao eixo de rotação do Encoder, que estava em contacto com a transmissão do motor, ou seja, o eixo do Encoder Incremental e o sensor rodariam a uma determinada velocidade aproximada, como se verifica nos esquemas representados nas Figuras 4.2, 4.3 e 4.4.

Transição(Pulso A)	Pulso B	Contagem
0	0	-1
0	1	1
1	0	1
1	1	-1

Figura 4.1: Funcionamento da codificação X2.

Nas medições com o Encoder Incremental foi utilizada a codificação X2 e a conversão de pulsos para [*Rad*] dada pela Equação 4.1,

$$Pos_{Encoder} = \frac{N_{Pulsos}}{2Npr} 2\pi \tag{4.1}$$

sendo $Pos_{Encoder}$ a posição angular do eixo do Encoder em [*Rad*], N_{pr} o número de pulsos por revolução e N_{Pulsos} o número de pulsos contados.

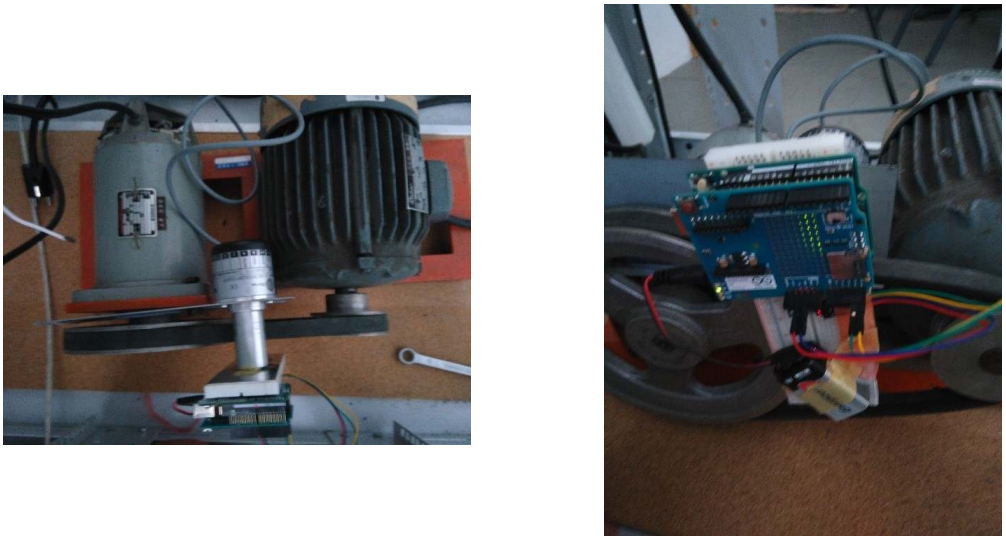


Figura 4.2: Encoder Incremental + Arduíno acoplado ao motor DC.

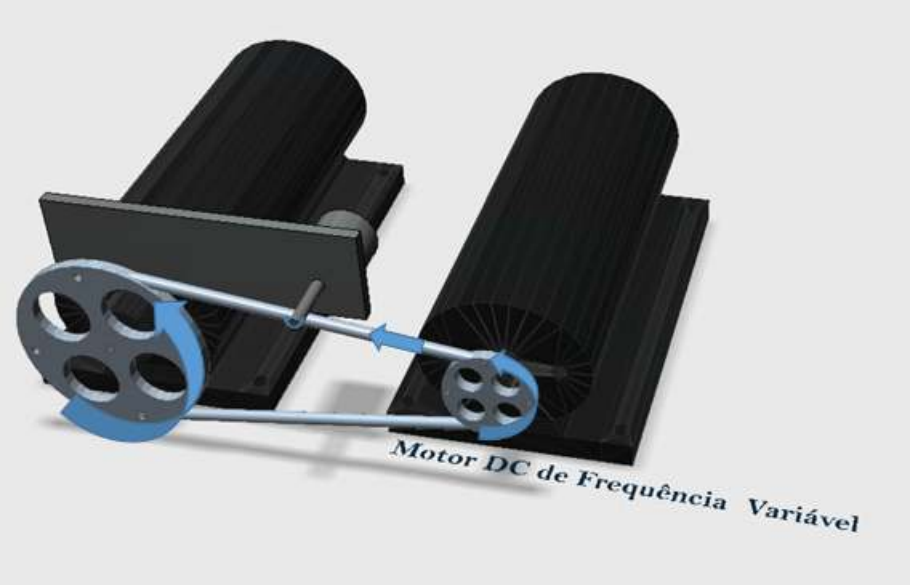


Figura 4.3: Esquema representativo da ligação entre os motores.

Figura 4.4: Esquema representativo do acoplamento da estrutura criada ao motor.

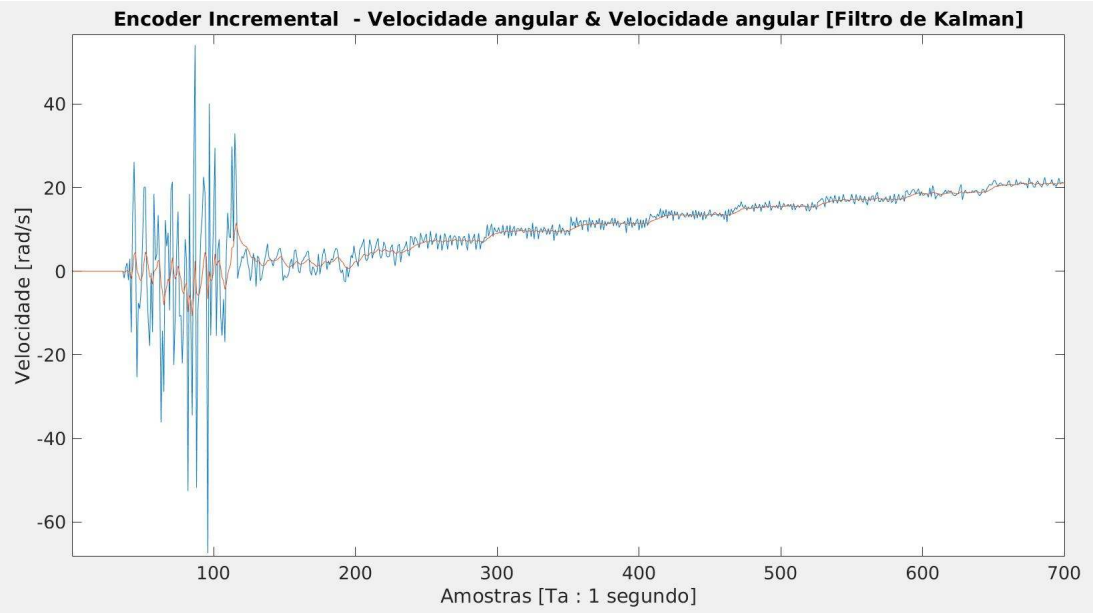


Figura 4.5: Resultados obtidos com a utilização do Encoder Incremental e a aplicação do Filtro de Kalman.

Velocidade Velocidade filtrada

Na Figura 4.5 podem ser observados os resultados obtidos com a utilização do Encoder Incremental, nos valores de frequência de $2Hz$ à $7Hz$ com um passo de $0,5Hz$ e com o tempo de amostragem de 1 segundo. Como o número de pulsos foi obtido de segundo em segundo, estes tinham como unidade $[Pulse/Sec]$. Após a sua conversão utilizando a Equação 4.1, o valor seria a quantidade de radianos percorrido num segundo.

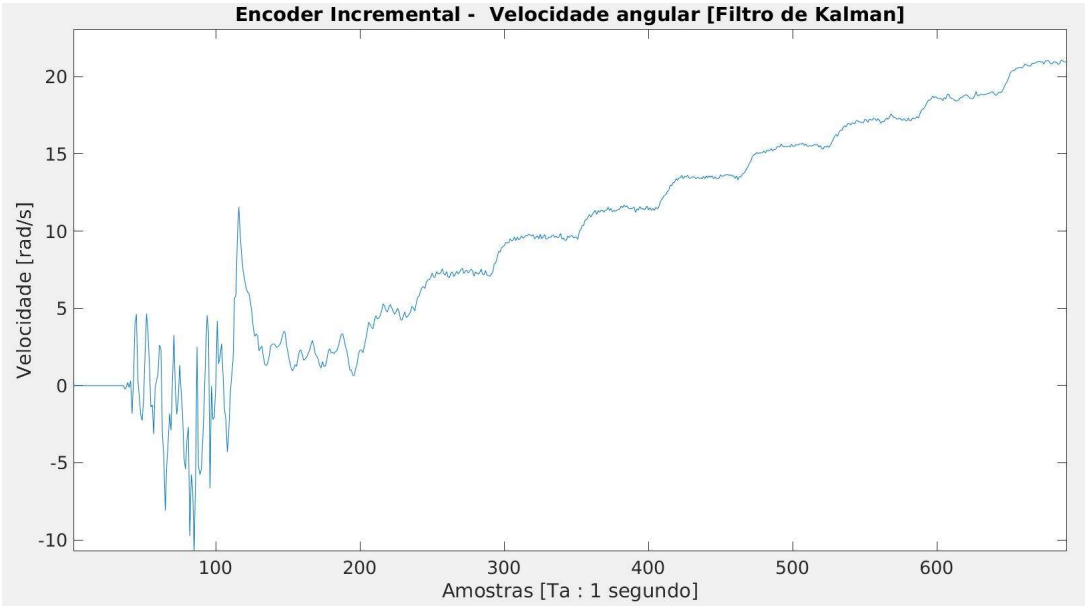


Figura 4.6: Dados obtidos do Encoder Incremental com o Filtro de Kalman.

Velocidade filtrada

Devido ao ruído apresentado pelo Encoder, principalmente na medição de baixas velocidades, foi necessária a implementação de um Filtro de Kalman com o intuito de filtrar as altas frequências e obter um conjunto de dados mais percetível, (Welch e Bishop, 2006), Figura 4.6. Para a implementação do Filtro de Kalman foram utilizados os parâmetros:

$R = 8,56$;

$Q = 0,2$;

sendo R o ruído de covariância na medição e Q o ruído de covariância do processo.

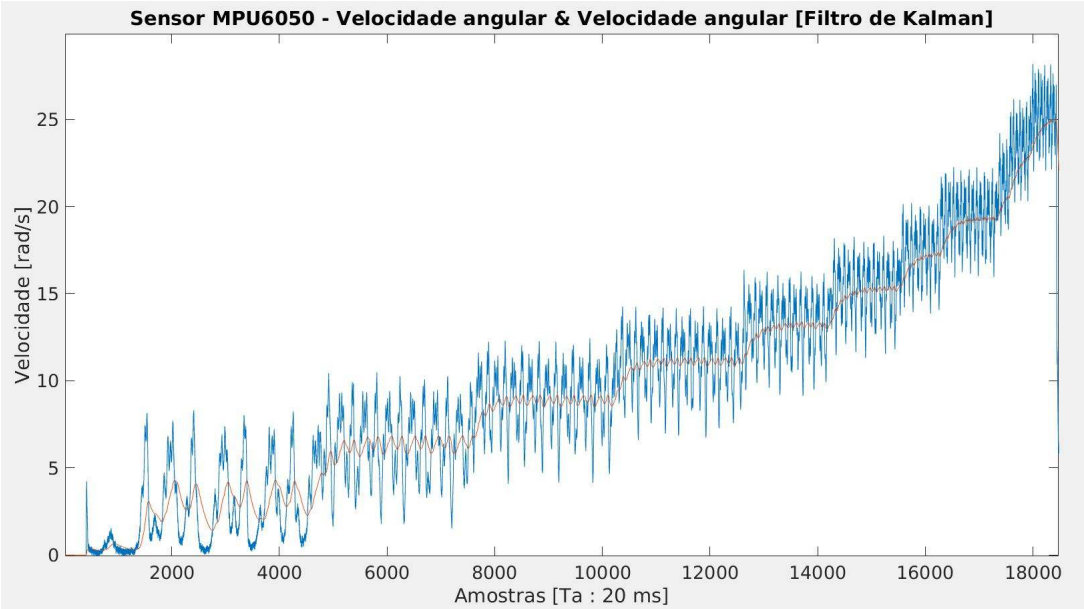


Figura 4.7: Dados obtidos do sensor MPU-6050 e a aplicação do Filtro de Kalman.
Velocidade Velocidade filtrada

Foi realizado exatamente o mesmo processo para o Sensor MPU-6050 para valores de frequência de $2Hz$ à $7Hz$, com um passo de $0,5Hz$ e com tempo de amostragem de $20ms$. Neste filtro foram utilizados os valores:

$R = 4$;

$Q = 0,001$.

Apesar do número de amostras ser diferente, o intuito foi, apenas, verificar que os valores de velocidade foram idênticos para cada uma das frequências aplicadas, o que neste caso informa também que o sensor reage melhor a velocidades mais baixas, Figuras 4.7 e 4.8, comparativamente a este Encoder Incremental que é tipicamente utilizado para medições de velocidades mais elevadas, (Eltra, 2014; Network e Republic, 2000).

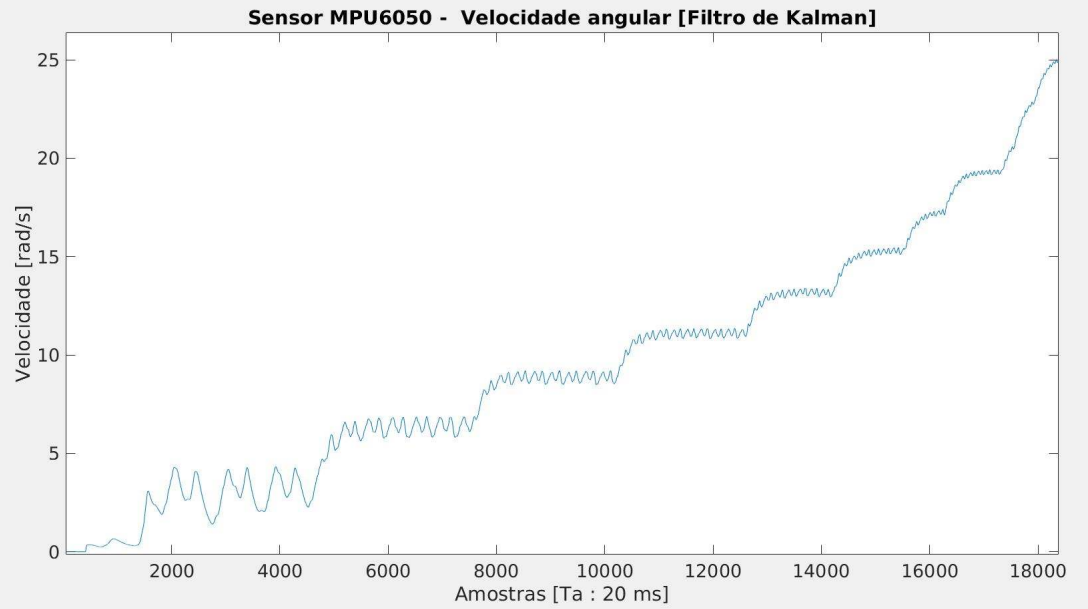


Figura 4.8: Dados obtidos do sensor MPU-6050 com o Filtro de Kalman.
Velocidade filtrada

Considerando o escorregamento existente entre a transmissão do motor e o eixo de rotação do Encoder, representado pelo esquema da Figura 4.4, os dois medidores sofreriam aproximadamente o mesmo escorregamento, devendo assim produzir resultados aproximados em velocidades intermédias.

Para terminar esta fase foi medida a velocidade em *rpm* com um Tacómetro. Trata-se de um terceiro método que tem como objetivo validar os anteriores. O Tacómetro foi encostado o mais próximo possível ao eixo de rotação, com o intuito de verificar se os valores do sensor MPU-6050 estavam corretos. Desta forma, o uso do Tacómetro podia implicar, também, algum escorregamento, o que influenciaria as medições a certas velocidades. Neste caso, foram realizadas medições dos $2Hz$ aos $6,5Hz$ com um passo de $0,5Hz$, Figura 4.9.

As velocidades associadas às frequências de $1Hz$ e $1,5Hz$ foram excluídas, devido à razão de transmissão entre o motor e o eixo do Encoder Incremental. A essas frequências, o eixo do Encoder apenas sofreria pequenas perturbações e caso se pressionasse o eixo de rotação com a ponta de medição do Tacómetro, esta causaria força suficiente para impedir a sua rotação. Como se pode observar na Figura 4.9, apesar do número de amostras retiradas, apenas em velocidades intermédias foi possível validar os outros métodos, uma vez que tanto em altas como em baixas velocidades os resultados da medição eram erróneos.

Mas porquê validar a velocidade medida pelo MPU-6050 quando o propósito é a posição angular? Sendo a velocidade validada e considerando intervalos de amostragem curtos, basta aplicar o fator temporal e obtém-se a posição angular. Visto que o sensor MPU-6050 originalmente media a velocidade angular, era mais fácil trabalhar com a unidade original e, posteriormente, converter para a unidade desejada.

Figura 4.9: Dados obtidos do Tacómetro.

Uma vez que em intervalos de tempo reduzidos (inferior a 1 segundo) pode-se assumir que o comportamento de um objeto é quase linear, a velocidade angular podia ser convertida para a posição angular média com a utilização da Equação 4.2,

$$\delta posicaoAngular = GYRO_Z \times t + GYRO_{Z_Anterior} \tag{4.2}$$

em que t é o tempo de amostragem escolhido e neste caso possui o valor de $20ms$.

Apesar da validação dos dados medidos por este sensor ter sido um sucesso, quando colocado em situações reais, nas quais existiam variações em mais do que um único eixo de rotação ao mesmo tempo e, em situações em que o tempo de aquisição de dados fossem horas ou dias, este sensor provava ser ineficiente. Dado que este não possui um magnetómetro, a posição angular é calculada tendo como referência o momento em que o Arduíno inicia e, por sua vez também o sensor. Ou seja, caso estivesse orientado para Sudeste, este seria o ponto de 0° e a soma dos valores de deslocação angular também teriam este ponto como referência. Para a aquisição de dados, este problema poderia ser resolvido caso o Arduíno fosse inicializado na direção de referência desejada (Norte), de forma a coincidir com a referência da estação meteorológica. No entanto, havia ainda o problema das inclinações, provocadas por pequenas perturbações segundo o eixo de *Pitch* e/ou de *Roll*, assim como o *drift* no *Yaw*, provocado pela constante integração dos valores com pequenos erros associados.

Do ponto de vista da medição do *Yaw*, caso o *Pitch* ou o *Roll* variassem, o eixo de rotação do *Yaw* também variaria e por este motivo a medição seria adulterada. Para acrescentar mais erros na medição, quando fossem calculadas as posições seguintes, estas estariam também adulteradas, criando a partir deste ponto um conjunto de valores de direção angular errados. Por este motivo foi utilizado o sensor AltIMU-10 v4 em conjunto com o Arduíno 1, que possuía um magnetómetro, permitindo assim solucionar os problemas acima expostos. Nesta fase, foi utilizado e adaptado o código disponibilizado

por (Github, 2017), "minimu-9-ahrs-arduino", que permitiu o cálculo dos valores do *Yaw*, do *Pitch* e do *Roll* com a compensação do *drift* no *Yaw*, com um tempo de amostragem de $20ms$. O código foi adaptado de forma a que, tanto o valor do *Yaw* como o valor do *Pitch* fossem recebidos e armazenados no cartão SD, juntamente com a data e a hora a que foram medidos, sendo posteriormente enviados (pela porta *Serial*) para o PC Central. Nessa adaptação, foi também adicionado o sistema de atualização da data e da hora do Arduíno 1, de acordo com o pacote de sincronismo recebido, Figura 3.6. Tanto o envio de dados para o PC Central como o sincronismo serão explicados detalhadamente na Secção 4.4.

Este sistema foi colocado dentro de uma caixa de instrumentação com Índice de Proteção 65(IP65), Figura 4.10, de forma a protege-lo de poeiras e da chuva, visto que esta seria acoplada ao aerogerador.

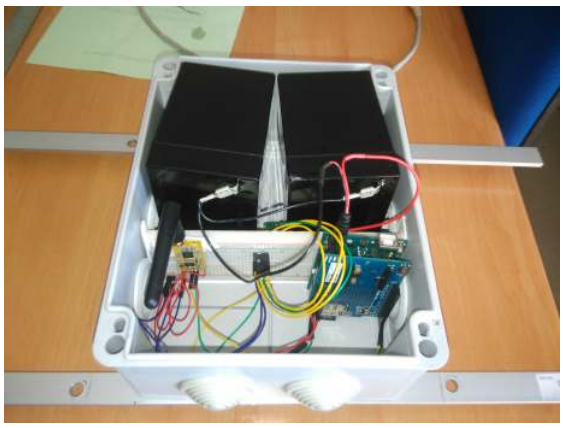


Figura 4.10: Caixa IP65 com as componentes utilizadas para a medição do ângulo de *Yaw*.

Esta caixa tinha no seu interior duas baterias de $12V$ e $5,5Ah$ ligadas em paralelo, de forma a aumentar a corrente elétrica, alimentando todo o sistema. Foi pendurada por correntes fixas ao teto, permitindo a interação com os diversos eixos de rotação.
Foram introduzidas grandes perturbações, manualmente, no eixo do *Pitch*, do *Roll* e do *Yaw*. Observou-se que apesar destas perturbações as medições do *Yaw* eram fidedignas, mesmo após diversas horas de integração, Figura 4.11.
É importante referir que o cálculo da direção do aerogerador foi realizado de forma a coincidir com a medição da direção do vento, sendo a direção incrementada quando este se movimentava no sentido inverso, sentido horário.

Antes da caixa ser acoplada ao aerogerador, foi estimado o número de dias que estas baterias alimentariam todo o sistema. Tendo em conta que o sistema de medição do ângulo de *Yaw* consumia no máximo $100mA$, era necessário perceber por quanto tempo as baterias conseguiriam fornecer esta corrente elétrica. Com este consumo, uma bateria ($5500mAh$) iria proporcionar um funcionamento durante 55 horas, ou seja, um total de 2 dias e 7 horas, Equação 4.3.

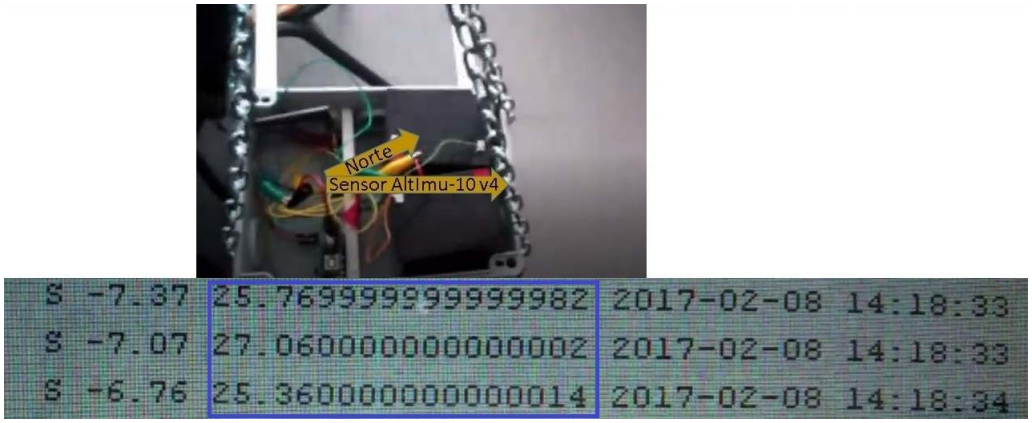


Figura 4.11: Validação dos dados obtidos pelo AltIMU-10 v4.

$$Dias = \frac{Capacidade_{Bateria}}{Consumo \times 24} \tag{4.3}$$

sendo $Dias$ o número de dias em funcionamento, $Capacidade_{Bateria}$ a capacidade da bateria em $[Ah]$ e $Consumo$ o consumo do sistema em $[A]$.

Uma vez que apenas uma das bateria aguentaria aproximadamente 2 dias e 7 horas, duas baterias iguais em paralelo iriam duplicar este tempo, ficando um total de 5 dias e 4 horas em funcionamento. Dada a sua localização, não era possível substituir as baterias facilmente, Figura 4.12, logo, o sistema deveria estar em funcionamento o máximo de tempo possível.

Na implementação deste sistema de aquisição de dados e na sua validação, foram utilizados os parâmetros apresentados na Tabela 4.1, sendo Ta o tempo de amostragem utilizado nos métodos/sensores de medição e validação.

Tabela 4.1: Parâmetros e métodos/sensores utilizados na implementação e validação deste sistema.

Parâmetros/Métodos	**MPU-6050**	**Encoder Incremental**	**AltIMU-10 v4**
Aquisição - Ta [segundos]	0,020	1	0,020
Filtro de Kalman - Parametro R	4	8,56	-
Filtro de Kalman - Parametro Q	0,001	0,2	-
Número de amostras adquiridas para validação	328892	705	4050
Intervalo de Frequências [Hz]	2-7	2-7	-

Figura 4.12: Caixa IP65 acoplada ao aerogerador.

4.2 Implementação do Sistema de Aquisição e Tratamento dos Dados da Estação Meteorológica

Nesta fase de implementação do sistema de aquisição e tratamento dos dados da estação meteorológica, o Arduíno 2 foi conectado de acordo com o esquema da Figura 3.8. Este recebia os pulsos associados à velocidade do vento e um número inteiro associado à direção do vento. A velocidade e a direção do vento foram calculadas com base na lógica descrita na Secção 3.4, Capítulo 3.

Uma vez que as medições da direção e da velocidade do vento possuem tempos de amostragem diferentes, foi necessário o uso de um *Interrupt* para realizar a contagem dos pulsos associados à velocidade e de um *Timer Interrupt* que, de 2,25 em 2,25 segundos concluía a contagem, disponibilizava a velocidade do vento e inicializava o contador de pulsos, (Cactus.io, 2016). É importante referir que, após uma adaptação do código original, este mesmo *Timer Interrupt* fez a gestão do tempo de amostragem da direção do vento, de forma a não se perder amostras, uma vez que este estava definido como 2,25 segundos apesar da direção possuir 1 segundo. Esta gestão de tempo de amostragem da direção do vento permitiu realizar o cálculo e a apresentação desta de 1 em 1 segundo e, aos 2,25 segundos apresentou, também, a velocidade do vento. Durante os 2,25 segundos a velocidade do vento era apresentada como sendo nula, uma vez que ainda estava a ser calculada.

Ao serem convertidos os valores inteiros da porta analógica 4 para *graus*(°), estes foram concatenados aos valores da velocidade do vento e aos dados relativos à data e à hora, para posteriormente serem enviados pelo módulo de comunicação RF. O funcionamento do Arduíno 2 na comunicação e relativamente ao sincronismo com os restantes sistemas será explicado na Secção 4.4.2.
Quando testado este sistema, foram obtidos os dados de velocidade do vento representados na Figura 4.13 e os da sua direção representados na Figura 4.14.

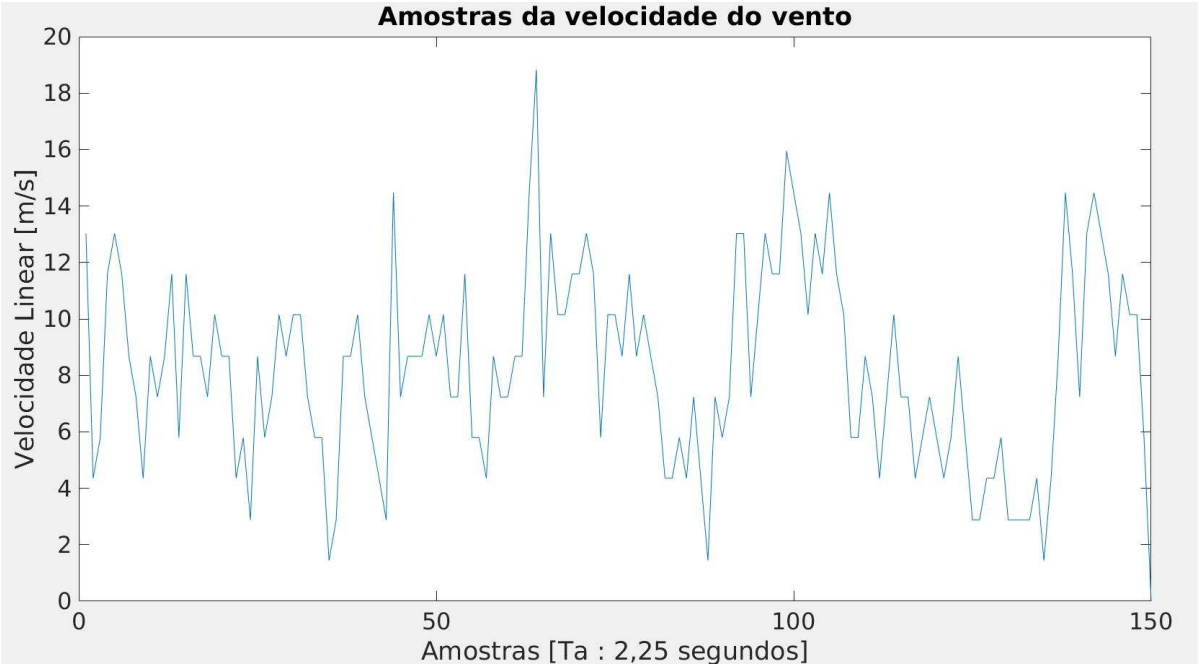


Figura 4.13: Velocidade do vento obtido.
Velocidade do vento

Para uma simples comparação foram utilizadas as plataformas online do Instituto Português do Mar e da Atmosfera, *www.ipma.pt/pt/*, do WindyTv, *www.windytv.com* e do WINDFINDER, *https://pt.windfinder.com/forecast/almada*, onde foi possível observar que a média dos valores obtidos era aproximada à demonstrada nas três plataformas.

Porém, uma vez que tanto a direção como a velocidade do vento variam com a rugosidade do terreno, os dados obtidos nas plataformas não poderiam ser iguais aos medidos, a não ser que os sensores de medição destas estivessem situados no mesmo local. Por este motivo, de forma a obter dados comparativos que se sujeitassem às mesmas condições, em cima do Departamento de Engenharia Electrotécnica e com os mesmos obstáculos, foi utilizado o Anemómetro *Heavy Duty Meter Series*, *EXTECH INSTRUMENTS*, para validação da velocidade do vento e, os resultados podem ser observados nas Figuras 4.15 e 4.16.

Ainda sobre a Figura 4.15, observa-se que durante os 2,25 segundos de medição da velocidade do vento, os valores adquiridos pelo sistema são nulos, uma vez que o processo de contagem dos pulsos do anemómetro ainda não tinha terminado.

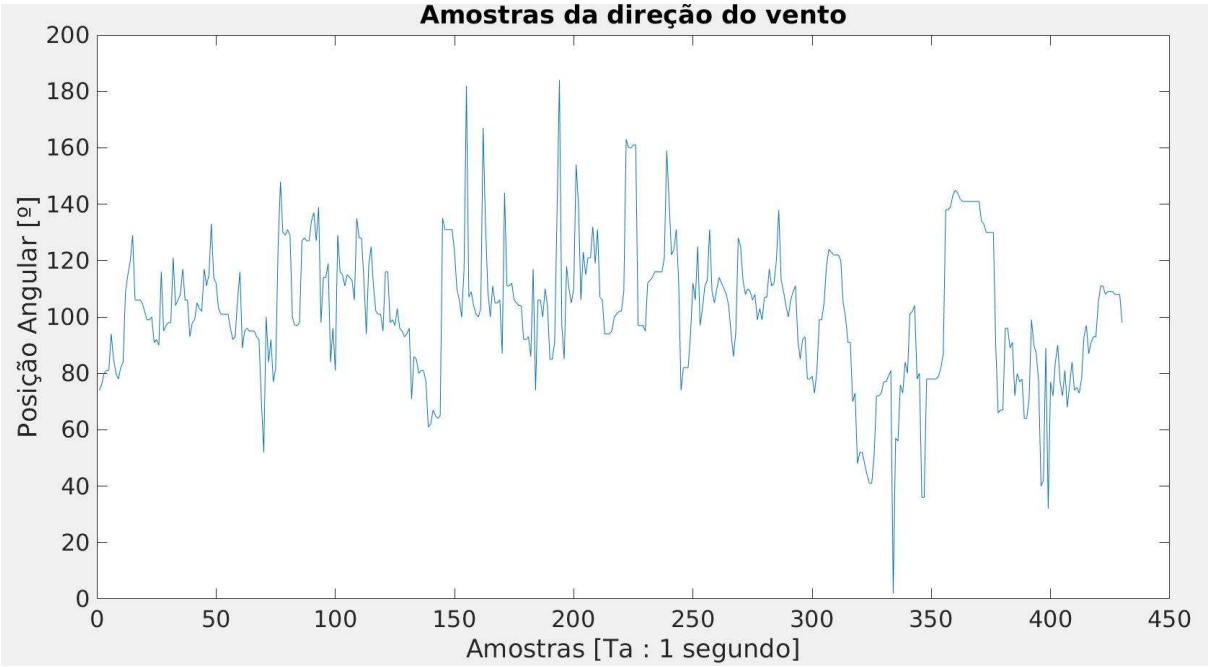


Figura 4.14: Direção do vento obtido.
Direção do vento

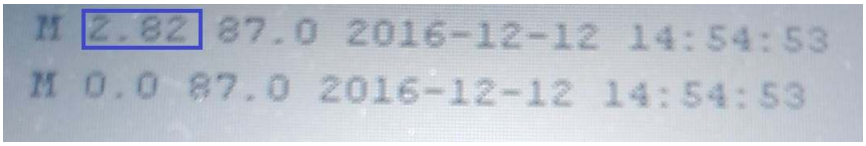


Figura 4.15: Velocidade do vento obtida pelo sistema de aquisição de dados da estação meteorológica.

Figura 4.16: Velocidade do vento medida com o Anemómetro *Heavy Duty Meter Series, EXTECH INSTRUMENTS*.

Relativamente à validação da direção do vento, esta foi realizada visualmente tendo como referência o norte magnético do planeta, pois, para a montagem deste material (Anemómetro de Davis), um dos requisitos é que o suporte do anemómetro esteja orientado para Norte, (Decagon Devices, 2016). Os resultados podem ser observados nas Figuras 4.17 e 4.18.

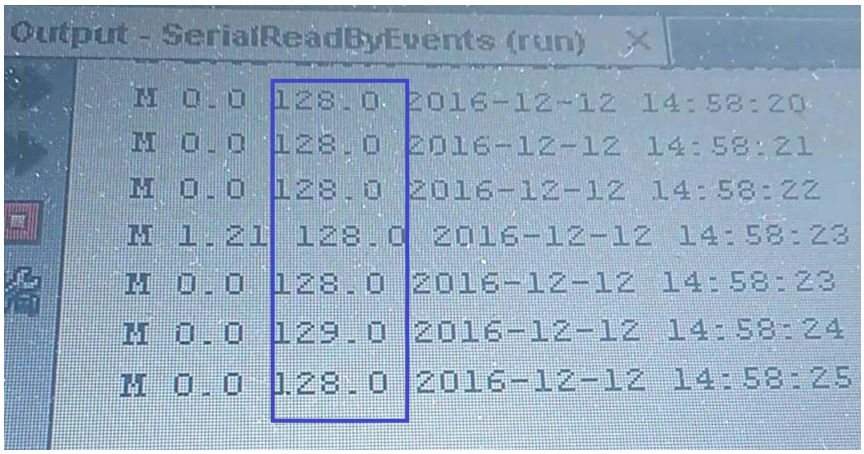


Figura 4.17: Direção do vento medida pelo sistema de aquisição de dados.

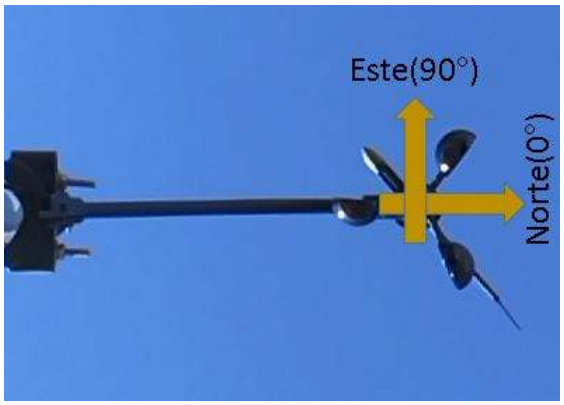


Figura 4.18: Direção do vento.

Como se pode observar os resultados eram bastante idênticos aos obtidos pelos métodos de validação utilizados, concluindo que os dados adquiridos são aproximadamente reais. Este sistema é representado pela Figura 4.19.

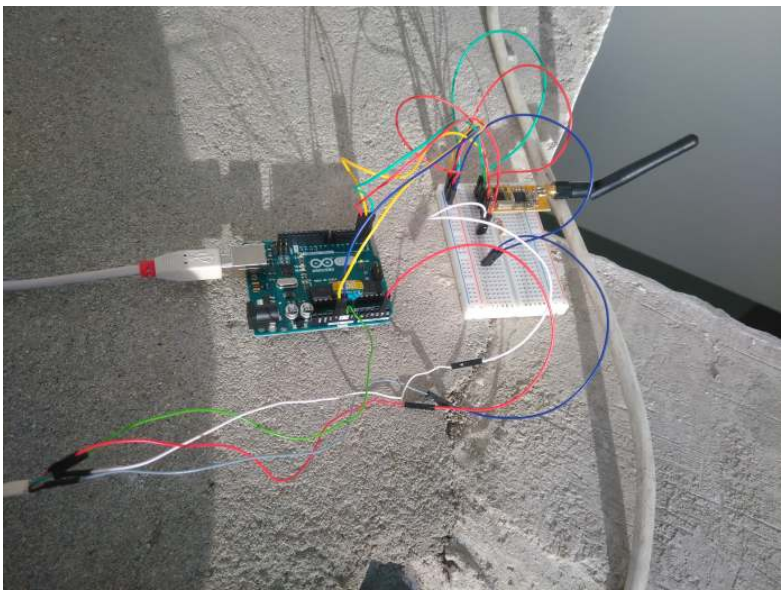


Figura 4.19: Sistema de aquisição de dados da velocidade e direção do vento.

4.3 Implementação do Sistema de Medição da Potência Elétrica Produzida

Nesta fase foi utilizado o Analisador de qualidade de energia, Fluke 43B, em conjunto com o Software FlukeView 3.34. O Fluke 43B foi utilizado, como representado na Figura 4.20, de forma a adquirir dados da produção elétrica.

Figura 4.20: Fluke 43B a realizar a medição da tensão e corrente elétrica produzida.

A aplicação FlukeView 3.34, como explicado no Capítulo 3 - Secção 3.5, permitia guardar diversos ficheiros com informação relativa à tensão e à corrente elétrica de aproximadamente 4 em 4 segundos, uma vez que o tempo de amostragem definido foi 0

segundos e, isto implicaria que a aplicação realizasse a aquisição de dados o mais rápido possível. Apesar de ser possível definir menos de 4 segundos, este era na prática o tempo de amostragem mínimo observado, mesmo quando definidos tempos de amostragem como 1 e 2 segundos.

Foi implementada em JAVA uma aplicação que lia os ficheiros criados pelo FlukeView 3.34 e organizava toda a informação, nomeadamente a data, a hora e os dados de produção elétrica (Figura 4.21), enviando posteriormente esses dados para o PC Central através de um APC220+ UART, tal como representado no esquema da Figura 3.11.

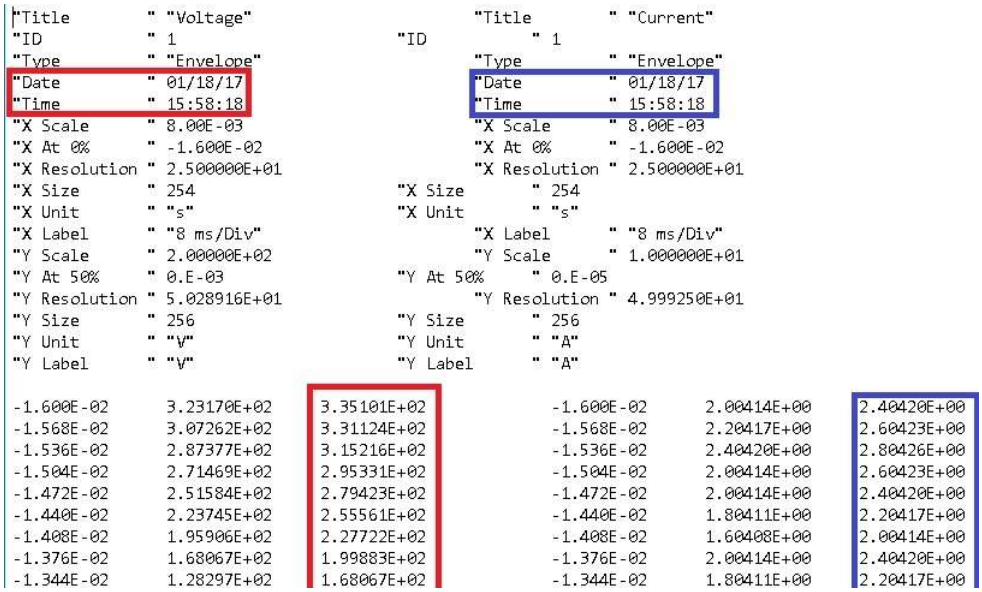


Figura 4.21: Ficheiro de texto criado pelo FlukeView 3.34, a vermelho estão representados os dados relativos a tensão elétrica e a azul os da corrente elétrica.

Apesar do tempo de amostragem ser 4 segundos, eram obtidos 256 valores associadas ao instante da amostra, ou seja, no instante $t = 1$ segundo obtinha-se um ficheiro com 256 amostras. No instante $t = 5$ segundos, 4 segundos depois, obtinha-se um segundo ficheiro com mais 256 amostras. Isso implicava que a leitura do Fluke 43*B* era descontinua pois não se possuía amostras para todos os segundos. Uma vez que se possuía 256 amostras num segundo, era necessário calcular a média desses valores instantâneos, de forma a se obter valores médios de tensão e valores médios de corrente elétrica e, por sua vez, calcular a potência instantânea naquele segundo. Como se observa no esquema da Figura 4.22, o primeiro passo foi verificar se o ficheiro que seria lido pela aplicação em JAVA, estaria a ser utilizado pelo FlukeView 3.34 (Exemplo: Ficheiro rec1.txt). Este processo foi realizado, verificando a existência de um segundo ficheiro criado também pelo FlukeView 3.34 (Exemplo : Ficheiro "rec2.txt"). Esta lógica, apenas, foi possível pois o FlukeView gera ficheiros de texto com o nome "rec" e um número que represente a gravação, como por exemplo "rec1.txt". Este número é incrementado de ficheiro para ficheiro, sendo o ficheiro "rec1.txt" o primeiro a ser criado e o "rec2.txt" o segundo, seguindo a lógica "rec['+*i*+'].txt" em JAVA, em que i é um número inteiro.

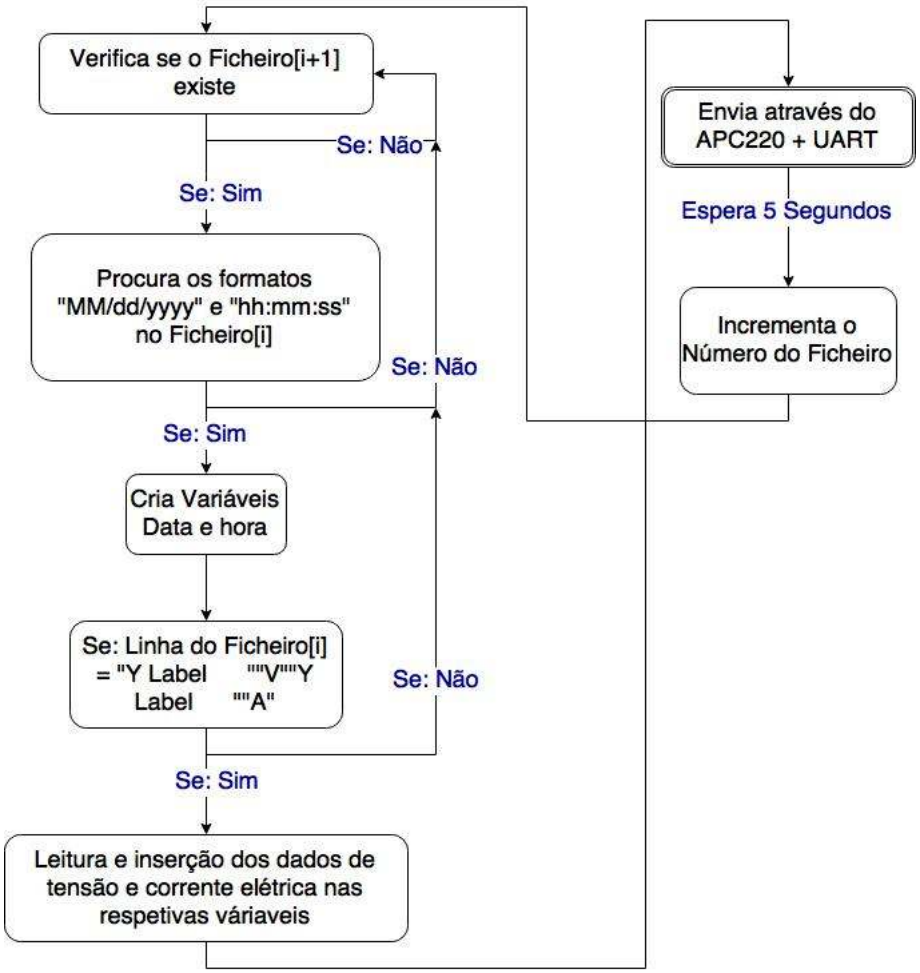


Figura 4.22: Esquema de funcionamento da aplicação JAVA responsável pela leitura dos ficheiro de texto.

Caso exista um ficheiro seguinte, a leitura inicia-se no anterior, começando por separar as linhas de texto em palavras, utilizando o comando *Split*. Ao serem obtidas as palavras de cada linha de texto, o objetivo seguinte foi encontrar os dados de data e de hora no ficheiro criado pelo FlukeView 3.34, criando assim variáveis de data e de hora em JAVA.

Após terem sido criadas as referidas variáveis (data e hora), o passo seguinte foi encontrar os dados da tensão e da corrente elétrica. Apesar de serem disponibilizadas três colunas de dados para a tensão elétrica e mais três para a corrente elétrica, foi contabilizada apenas uma fase, representada pela única coluna que possuía os valores máximos e mínimos de tensão e corrente elétrica descritos no ficheiro, ou seja, a terceira coluna de ambas as unidades elétricas. Este tipo de implementação foi possível, devido ao facto do FlukeView 3.34 manter sempre o mesmo formato nos seus ficheiros de texto.

Quanto ao sincronismo necessário entre todos os sistemas de aquisição e transmissão dos dados para o PC Central, Figura 3.11, este será explicado na Secção 4.4.

4.4 Comunicação e Sincronismo

Após terem sido validados os sistemas de aquisição de dados, foi criado um sistema de comunicação e sincronismo entre estes e o PC Central. Este processo baseou-se na utilização de dois módulos de comunicação RF, o Módulo RF 433MHz e o Módulo APC220, tendo sido escolhido o que melhor cumpria as necessidades de comunicação e sincronismo.

4.4.1 Módulo RF 433MHz

Quando utilizado o Módulo RF 433MHz, a comunicação e sincronismo deveriam ser realizados segundo o esquema da Figura 3.14. Porém, foi testado previamente o módulo como se pode observar no esquema da Figura 3.16. Segundo este, o Arduíno 1 seria o transmissor e o Arduíno 4 o recetor, tendo estes sido programados com base no código disponibilizado pelo "FILIPEFLOP " (Capítulo 3 - Secção 3.6).
Do código do referido autor, foi utilizada a função "send",

```
1  void send(char *message) {
2    vw_send((uint8_t *)message, strlen(message));
3    vw_wait_tx(); // Aguarda o envio de dados
4  }
```

cujo objetivo era, enviar a mensagem guardada num apontador para um *array* de *char* com o nome "message". Para a receção, foi realizada uma adaptação do código do referido autor.

```
1  uint8_t message[VW_MAX_MESSAGE_LEN];
2  uint8_t msgLength = VW_MAX_MESSAGE_LEN;
3  if (vw_get_message(message, &msgLength)) // Non-blocking
4  {
5    Serial.print("Recebido: "); // Imprime os dados recebidos
6    for (int i = 0; i < msgLength; i++)
7    {
8      Serial.write(message[i]);
9    }
10   Serial.println();
11 }
```

Ao ser implementada a montagem representada na Figura 4.23, observou-se, como relata a Figura 4.24, a receção dos dados com algum atraso, inclusive a perda de alguns dados devido à qualidade de comunicação, à distância, à alimentação elétrica e ao meio em que se encontravam os módulos RF.

Por outro lado, o transmissor necessitava de enviar os dados relativos à hora e à data, de forma a manter o segundo Arduíno sincronizado. Para tal, foi utilizada a biblioteca *TimeLib* para o Arduíno Uno, que possuía diversas funções preparadas para o sincronismo.

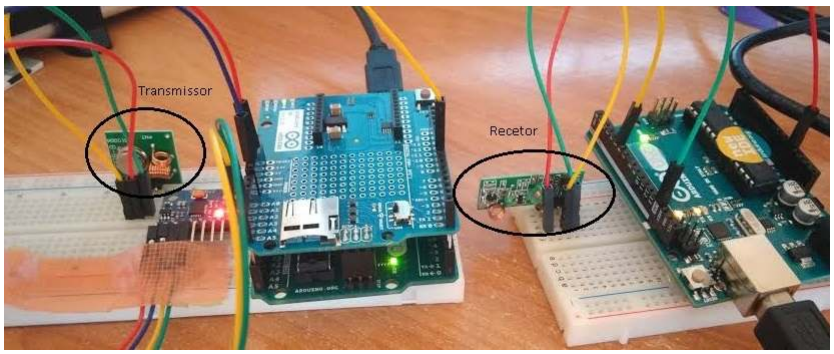


Figura 4.23: Módulo RF Transmissor e Recetor ligado a 2 Arduínos Uno.

COM3 (Arduino Uno)

Recebido:	-0.00	11.00
Recebido:	-0.00	10.99
Recebido:	0.00	10.98
Recebido:	0.00	10.98
Recebido:	-0.00	10.97
Recebido:	-0.00	10.94
Recebido:	-0.00	10.93
Recebido:	0.00	10.93
Recebido:	-0.00	10.93
Recebido:	-0.00	10.91
Recebido:	-0.00	10.91
Recebido:	-0.00	10.90
Recebido:	-4.51	325.89
Recebido:	-3.09	304.10
Recebido:	-1.05	296.74
Recebido:	-0.51	293.18
Recebido:	-0.00	293.16
Recebido:	-0.10	292.49

COM4 (Arduino Uno)

-0.00	11.00
-0.00	10.99
-0.00	10.98
0.00	10.98
0.00	10.98
0.00	10.98
-0.00	10.97
-0.00	10.94
-0.00	10.93
0.00	10.93
-0.00	10.93
-0.00	10.91
-0.00	10.91
-0.00	10.90
-2.03	357.39
-4.51	325.89
-3.09	304.10
-1.05	296.74

Figura 4.24: Envio e receção de dados por parte do módulo RF transmissor e recetor.

Com base nos exemplos fornecidos pela referida biblioteca, optou-se por seguir aproximadamente a mesma abordagem nesta fase inicial, que seria introduzir manualmente a hora e a data pretendida para o Arduíno. Partindo do princípio que o Arduíno 1 e 4 estariam em comunicação com um computador e, este por sua vez ligado a uma rede Internet, mesmo que o computador deixasse de estar sincronizado com a rede, o Arduíno iria manter a hora e a data introduzidas manualmente. No caso desta biblioteca, era mais simples introduzir os dados em *Epoch*, ou seja, um número *long* que representa a hora/-minutos/segundos do dia/mês/ano desejada. Este número é, de uma forma simplificada, o número de segundos desde o dia 1 de Janeiro de 1970 às 00 : 00 : 00 horas até à data e à hora introduzida. Existem diversos conversores online de hora/data para *Epoch*, portanto foi possível utilizar a abordagem inicial do exemplo fornecido pela biblioteca, sendo que no transmissor (Arduíno 1), foi pedida a hora/data no formato de *Epoch* e foi recebida segundo o seguinte código.

```
if (Serial.available() > 1) { // wait for at least two characters
  char c = Serial.read();
  if ( c == TIME_HEADER) {
    processSyncMessage();
  }
  else if ( c == FORMAT_HEADER) {
    processFormatMessage();
  }
}
```

No presente código, a função *processSyncMessage()* é responsável por definir a hora e a data do Arduíno e, a função *processFormatMessage()* serve para mudar o formato da data. De forma a não confundir os dados recolhidos e tratados pelo Arduíno 1 com os dados do sincronismo, quando enviados através do módulo RF transmissor, estes (hora/data) possuíam um caractere *D* antes da mensagem e eram enviados de 1 em 1 minuto.

```

void processSyncMessage() {
  unsigned long pctime;
  const unsigned long DEFAULT_TIME = 1357041600;
  pctime = Serial.parseInt();
  if ( pctime >= DEFAULT_TIME) {
    // check the integer is a valid time (greater than Jan 1 2013)
    setTime(pctime);// Sync Arduino clock to the time received
    //on the serial port
  }
}
```

No Arduíno recetor (Arduíno 4) os dados recebidos eram verificados. Caso não possuíssem o caractere *D*, seriam considerados dados informativos e deveriam ser mostrados. Caso possuíssem um caractere *D* no inicio da *String*, eram divididos pelas respetivas variáveis de hora e data e, estas atualizariam a data/hora do Arduíno recetor.

Ao serem recebidos e distribuídos os dados de sincronismo, a função *setTime* da biblioteca definia a data/hora do Arduíno 4. Não houve verificação de sincronismo ou falta do mesmo, simplesmente de 1 em 1 minuto, foram atualizadas a hora e a data de forma a que, quando os dados fossem analisados, se soubesse exatamente a que horas/minutos/-segundos foram obtidos, visto o relógio ser o mesmo.

Apesar do sistema de sincronismo ter sido implementado com sucesso, os Arduínos 1 e 2 precisariam de enviar os respetivos dados adquiridos e, de 1 em 1 minuto, receberem os pacotes de sincronismo vindos do Arduíno 4 (Figura 3.14). Por sua vez, o Arduíno 4 necessitaria de receber os dados enviados pelos Arduínos 1, 2 e 3 e, enviar os pacotes de

sincronismo. Ou seja, os Arduínos 1, 2 e 4 necessitariam tanto de enviar como de receber e, por isso, os sensores de comunicação tinham que ser *Transceivers*.

Com isto, chegou-se à conclusão que, trabalhando com estes módulos, seriam necessários um total de sete Arduínos Uno, uma vez que apenas um não conseguiria enviar e receber dados utilizando um módulo transmissor e um módulo recetor em separado. Este problema motivou o uso de *Transceivers*.

4.4.2 Módulo RF APC220

Considerando os problemas associados ao Módulo RF433Mhz, foi utilizado o Módulo RF APC220, pois é um *Transceiver* e cumpre as necessidades de comunicação descritos anteriormente.

Antes de qualquer uso destes módulos é necessário configurá-los de forma a que possuam a mesma *RF frequency*, *TRx Rate*, *NET ID*, *NODE ID*, *RF Power* e *Serial Rate*. Para essa configuração foi utilizado o *RF-Magic (for APC22x V1.2A)*, sendo as caraterísticas definidas representadas na Figura 4.25.

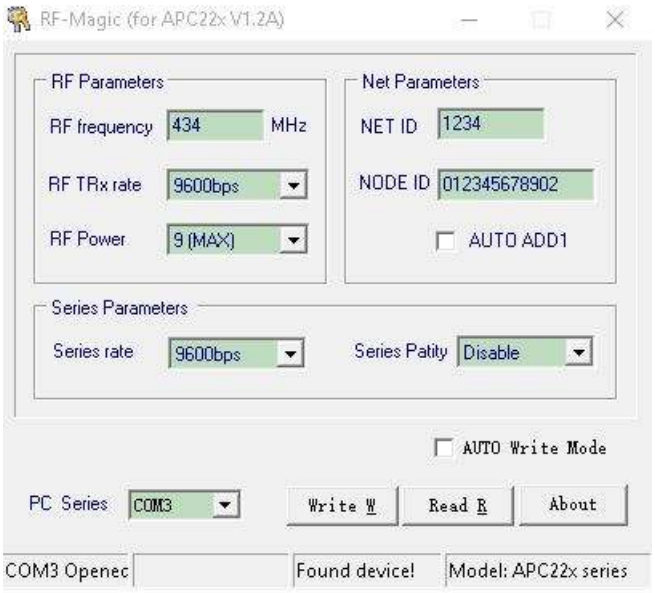


Figura 4.25: Configuração do Módulo RF APC220, *RF-Magic (for APC22x V1.2A)*.

Após a fase de configuração, passou-se à fase de implementação do sistema de comunicação entre os diversos Arduínos, utilizando este novo módulo de comunicação RF.

Com a utilização deste módulo, a arquitetura de aquisição de dados foi modificada (Figura 3.15, Capítulo 3 - Secção 3.6), uma vez que se tornou possível a utilização da interface UART/TTL em vez de um Arduíno Uno ligado a um computador.

Após a modificação da referida arquitetura, a programação associada a cada um dos Arduínos utilizados anteriormente para efeitos de teste, tornou-se desnecessária. Com a utilização deste módulo de comunicação RF, não era necessária nenhuma função específica para transmitir ou receber dados, mas sim, uma simples leitura e escrita na porta

Serial. Nesta fase, foi desenvolvida uma aplicação em JAVA para leitura e escrita para a porta *Serial*, representada nos esquemas de aquisição de dados da direção do aerogerador, da estação meteorológica e da produção eólica, pelo mesma simbologia de JAVA, Figuras 3.6 3.7 e 3.11.

4.4.2.1 Comunicação e Sincronismo - Arduíno 1

No Arduíno 1, após a leitura dos dados relativos ao aerogerador, estes foram inseridos numa String juntamente com a hora e a data de aquisição dos mesmos e, posteriormente enviados pela porta *Serial* para o PC Central de 240 em 240*ms*. O pacote enviado possuía os campos representados pela Figura 4.26.

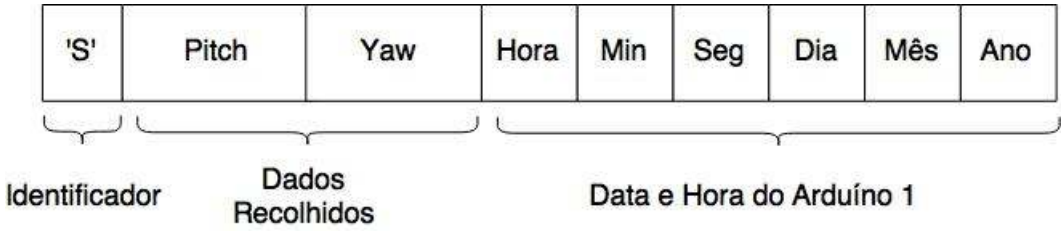


Figura 4.26: Campos do pacote de dados enviado pelo Arduíno 1.

Para além da funcionalidade descrita, sempre que o mesmo recebesse um pacote com o identificador "D", atualizaria a sua data e hora de acordo com os dados recebidos nesse pacote de sincronismo e fechava o ficheiro situado no cartão SD, permitindo assim a gravação deste (Anexo C.1).

4.4.2.2 Comunicação e Sincronismo - Arduíno 2

No caso do Arduíno 2, este calculava a velocidade e a direção do vento com base nos valores recebidos da estação meteorológica (Secção 4.2), concatenava esses dados numa *String* juntamente com a informação da data e da hora da sua aquisição (Figura 4.27) e, em seguida, enviava-os para o PC Central de 1 em 1 segundo.

Figura 4.27: Campos do pacote de dados enviado pelo Arduíno 2.

Quanto à receção dos dados de sincronismo, esta foi realizada de forma idêntica à do Arduíno 1, pois o pacote de sincronismo utilizado era igual para qualquer um dos Arduínos (Anexo C.2).

4.4.2.3 Comunicação e Sincronismo - PC2

O PC2 recebia os dados da produção de energia elétrica produzida através do Analisador de Qualidade de Energia. Por sua vez, a aplicação criada em JAVA realizava o envio dos dados recolhidos através do APC220 de 5 em 5 segundos, utilizando a porta *Serial* de forma idêntica ao realizado nos Arduínos. Os pacotes de dados enviados possuíam os campos representados na Figura 4.28.

Figura 4.28: Campos do pacote de dados enviado pelo sistema de aquisição dos dados de Potência elétrica.

Relativamente ao sincronismo, este foi realizado pelo PC2, ao estar ligado à mesma rede Internet que o PC Central (Anexo C.3), responsável pelo sincronismo dos Arduínos. Relativamente ao Fluke 43B, este foi configurado inicialmente com a mesma hora e data que o PC2.

4.4.2.4 Receção dos dados de Sincronismo nos Arduínos Uno

Quanto à leitura da porta *Serial*, esta era realizada sempre que houvesse algum pacote para ser lido em cada um dos Arduínos, uma vez que os pacotes de sincronismo eram enviados pelo PC Central de minuto a minuto, porém, dependendo dos fatores que influenciassem a comunicação, estes poderiam chegar aos diversos Arduínos em instantes diferentes.

Após a leitura da porta *Serial*, era verificada a origem do pacote recebido a partir do caractere na posição 1 da String, que neste caso em específico, era verificado se o pacote possuía um *D* como primeiro caractere. Caso possuísse, este pacote traria dados de data e hora do PC Central.

```
1 // Leitura da porta serial e sincronização
2 while(Serial.available()>0){
3  if(Serial.read()=='D'){
4    D=Serial.readString();
5    pctime=D.toInt();
6    setTime(pctime);
7   }
8  }
```

Apenas os Arduínos Uno validavam os pacotes começados por "D", Figura 4.29, extraindo dele a data/hora e definindo como sendo a sua data e a hora atual.

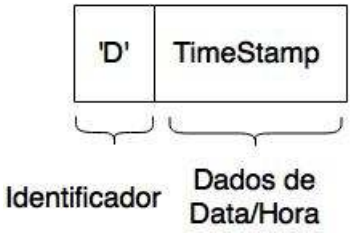


Figura 4.29: Campos do pacote de dados enviado pelo "APC + UART/TTL".

Este processo criava um sincronismo entre os Arduínos e o PC Central, pois o pacote era único e tinha sido enviado por *Broadcast*. Apesar de haver algum atraso associado à distância, ao meio de propagação da onda e até mesmo ao clima, este atraso era mínimo (2 a 3 segundos) e, sendo conhecido, poderia ser acertado.

4.4.2.5 Comunicação e Sincronismo - PC Central

O Módulo RF APC220 acoplado ao UART/TTL foi ligado diretamente ao PC Central por uma porta USB que, após a instalação dos seus *drivers*, adquiriu uma porta *Serial* associada, Figura 4.30 e Figura 4.31. Por sua vez, essa mesma porta *Serial* foi utilizada para receção e transmissão de dados, fazendo o trabalho dos Arduíno 4 e 4.1 no caso do Módulo RF 433Mhz.

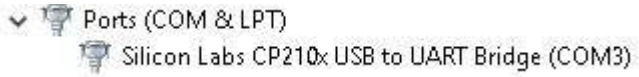


Figura 4.30: Porta *Serial* associada ao UART/TTL.

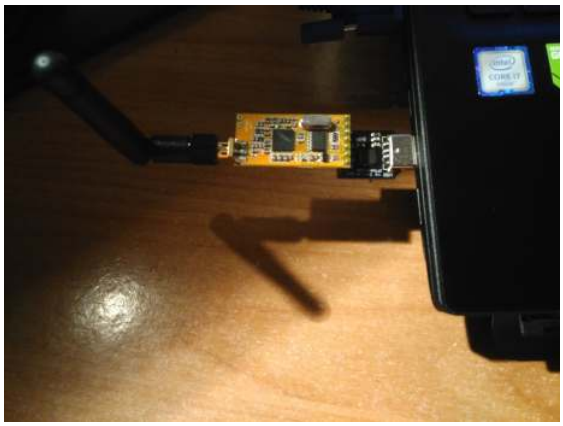


Figura 4.31: UART/TTL ligado a porta USB do PC Central.

Foi utilizada a estrutura de comunicação *Serial* nos dois sentidos, (RXTX, 2016), onde já tinham sido programadas duas *Threads* em JAVA para receção e transmissão de dados. Essas *Threads*, sendo uma para receção e outra para transmissão, serviam para que a transmissão funcionasse de forma independente da receção, ou seja, em paralelo. Para que fosse possível comunicar com o *UART/TTL* foi necessário, para além da inserção dos

parâmetros da porta *Serial*, *DATABIT*, *PARITY*, *STOBITS* e do *BitRate*, configurar também o *DTR* e o *RTS*.

```
SerialPort serialPort = (SerialPort) commPort;
serialPort.setSerialPortParams(9600,SerialPort.DATABITS_8,
SerialPort.STOPBITS_1,SerialPort.PARITY_NONE);
serialPort.setDTR(true);
serialPort.setRTS(false);

InputStream in = serialPort.getInputStream();
OutputStream out = serialPort.getOutputStream();

(new Thread(new SerialReader(in))).start();
(new Thread(new SerialWriter(out))).start();
```

Tendo sido configurada a comunicação com a porta *Serial*, passou-se à implementação da receção dos dados. Ao receber pacotes, era verificado o seu identificador que indicava de onde vinha e que tipo de dados possuía:

- S - Pacote enviado pelo Arduíno 1 com os dados do aerogerador;
- M - Pacote enviado pelo Arduíno 2, acoplado à estação Meteorológica;
- P - Pacote enviado pelo PC2, responsável pela medição da potência produzida;
- D - Pacote enviado pelo PC Central, responsável pelo sincronismo.

Após essa validação os dados eram tratados como será explicado na Secção 4.5.

Na transmissão foi realizada uma leitura da data e da hora do PC central, que por sua vez estava síncrona com a rede Internet. A data/hora obtida foi enviada de 60 em 60 segundos para a porta *Serial* com "D" como a identificação do pacote.

Na Tabela 4.2 estão representados os intervalos de tempo associados à transmissão de pacotes de dados e de sincronismo de cada um dos sistemas.

Tabela 4.2: Intervalos de tempo na transmissão de pacotes de dados/sincronismo

Sistemas/Transmissão de pacotes	**Pacotes de dados [Segundos]**	**Pacotes de sincronismo [Segundos]**
Arduíno 1	0,240	-
Arduíno 2	1	-
PC2	4	-
PC Central	-	60

4.5 PC Central e Plataforma de Monitorização dos Dados

O PC Central era a unidade responsável por receber os dados dos diferentes sistemas de aquisição (Arduínos 1 e 2 e, PC2), por processá-los e guardá-los numa base de dados MySQL em tempo real. Quanto à Plataforma de monitorização dos dados, esta situava-se também no PC Central e, foi implementada com o propósito de facilitar o acesso e a análise dos dados recebidos dos diversos sistemas de aquisição. Esta acedia aos dados que tinham sido préviamente inseridos na base de dados MySQL, Figura 4.32, e apresentava-os de forma gráfica e síncrona.

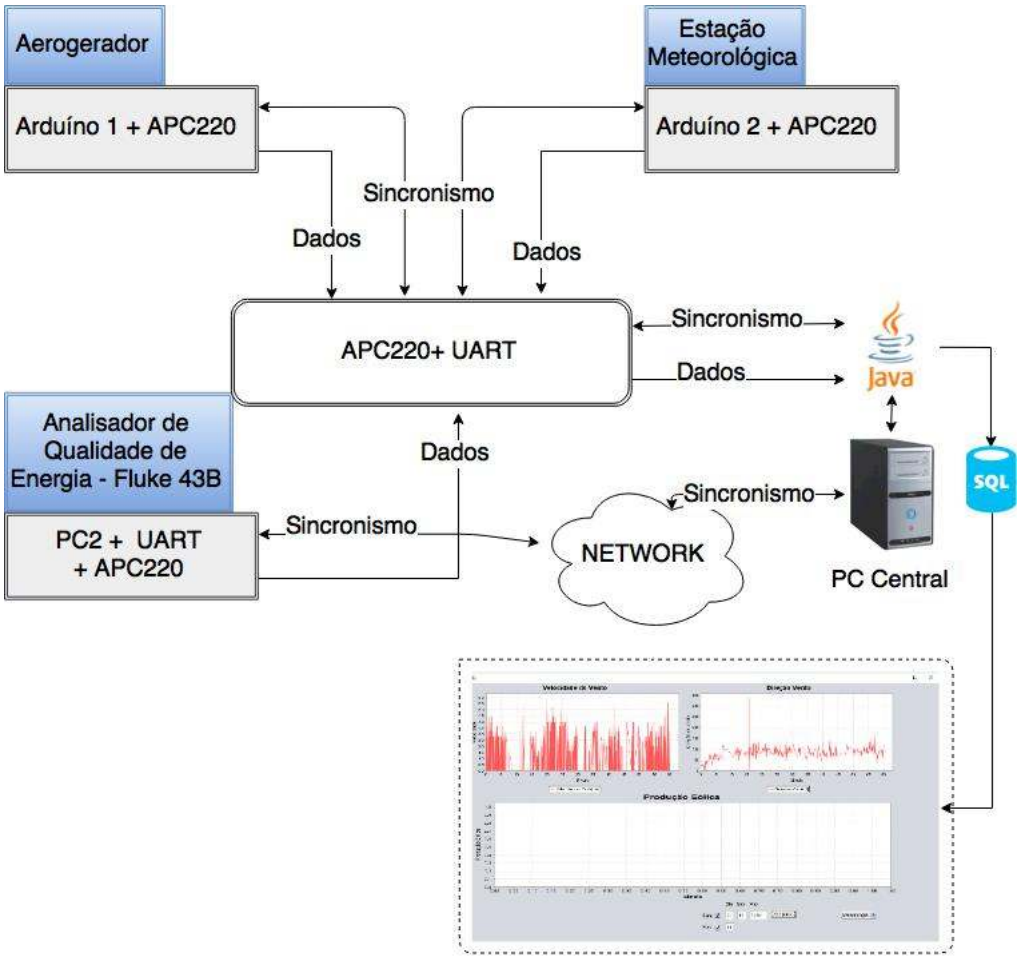


Figura 4.32: Esquema de aquisição, sincronismo, inserção e plataforma de monitorização dos dados.

4.5.1 PC Central

Na fase inicial do processo de criação da base de dados foi utilizado o *MySQL database server 5.7.15*, onde foram definidos os utilizadores com acesso a esta. A base de dados foi criada seguindo o tutorial disponibilizado pela plataforma *NetBeans IDE 8.0.2*, (NetBeans, 2016), utilizando os seguintes dados :

- Username: ThesisDB;
- Password: thesis;
- Host: localhost;
- role: DB Admin;
- Windows Service Name: MySQL57;
- Port: 3306;
- User Root: root;
- Password Root: thesis123.

Após a criação da base de dados e dos seus utilizadores, foi necessário programar o acesso a esta, de forma a que os dados fossem lidos através da porta *Serial COM3* e introduzidos em tempo real. Para a leitura da porta *Serial* foi utilizada a *Thread* de leitura (citada na Secção 4.4) e adaptada de forma a que não fosse uma simples leitura da porta, mas sim, um *Insert* dos dados na base de dados.

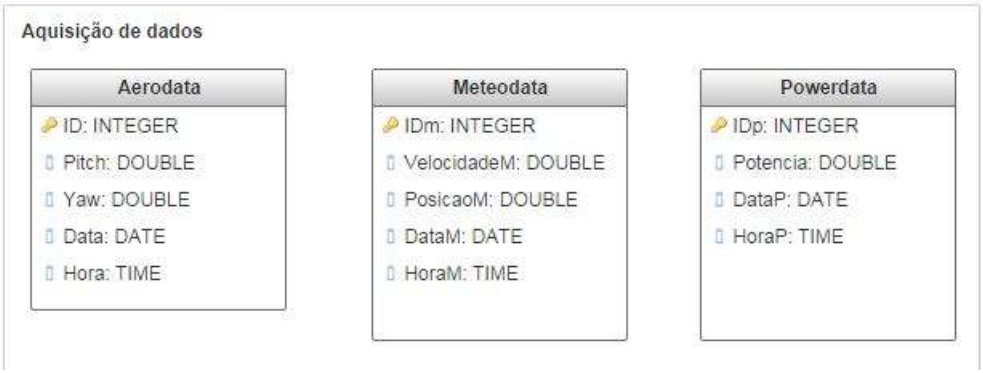


Figura 4.33: Tabelas e campos da base de dados utilizados pela plataforma.

Os dados enviados pelos diversos sistemas de aquisição foram extraídos da *String* com o comando *Split* e introduzidos nos campos representados na Figura 4.33, utilizando para comunicação, criação de acesso e escrita na base de dados a seguinte codificação,

```
Class.forName("com.mysql.jdbc.Driver");
conn = DriverManager.getConnection(DB_URL, USER, PASS); // Login
stmt = conn.createStatement(); // Cria o acesso a base de dados

// Executa a inserção ou update descrito pela String sql na base de dados
int rs = stmt.executeUpdate(sql);
stmt.close();
conn.close();
```

sendo os dados associados ao *Login* representado por,

```
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost:3306/mysql";
static final String USER = "root";
static final String PASS = "thesis123";
```

Quanto aos dados recebidos, estes foram extraídos do *buffer* e inseridos na base de dados de acordo com a sua identificação ("S", "M" ou "P"), Anexo A.

4.5.2 Plataforma de Monitorização de Dados

Terminada a fase de criação e inserção dos dados recebidos na base de dados, foi implementada uma interface que facilitaria o acesso e a análise do conjunto de dados armazenados previamente, a plataforma de monitorização, Anexo B. Para a implementação, foi criada em JAVA um *JFrame Form* onde foram introduzidas as diversas *jLabel*, *jButton*, *jCheckBox* e *jTextField*, Figura 4.34, cujo funcionamento está representado na Máquina de estados exposta na Figura 4.35.

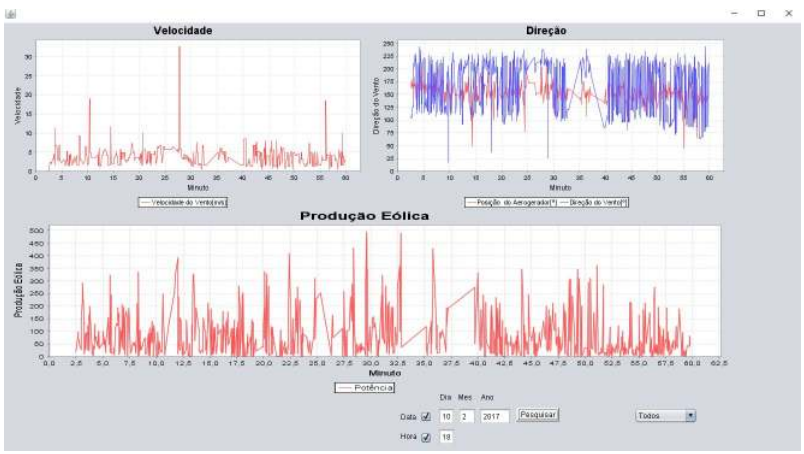


Figura 4.34: Plataforma de monitorização da Base de Dados.

O funcionamento pode, também, ser visualizado ao analisar o código presente no botão "Pesquisar", onde foi implementada a lógica de funcionamento da pesquisa. Caso seja realizada a pesquisa completa, opção 3, esta apresenta os campos "Pitch", "Yaw", "VelocidadeM", "PosicaoM" e "Potencia" tendo como fator comum a data e a hora, representado pela Figura 4.36. Para obter esse resultado foi utilizada a seguinte *Query*.

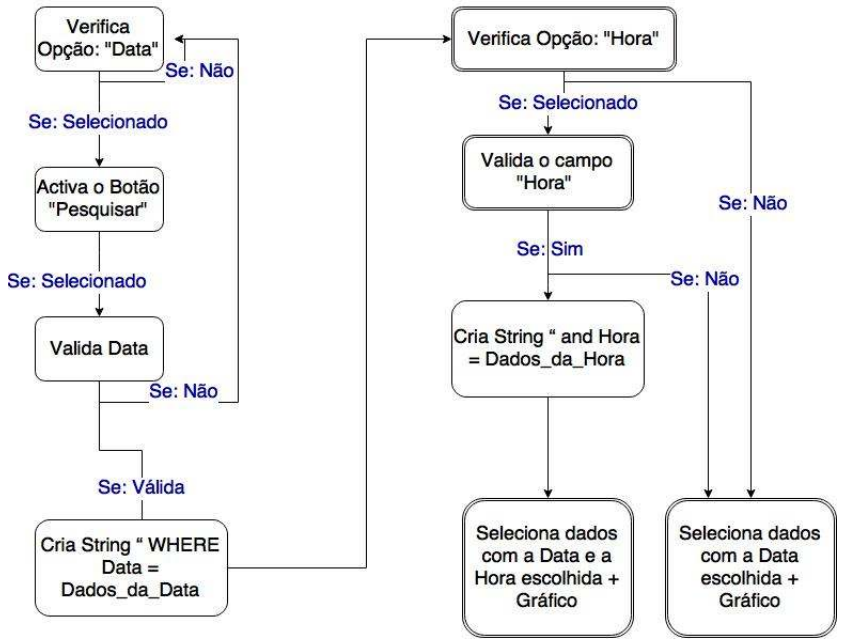


Figura 4.35: Máquina de Estados - Funcionamento geral da plataforma de monitorização.

Figura 4.36: Campos das tabelas "Aerodata", "Meteodata" e "Powerdata" selecionados na pesquisa completa.

```
SELECT
mysql.aerodata.Pitch,mysql.aerodata.Yaw,mysql.meteodata.VelocidadeM,
mysql.meteodata.PosicaoM,mysql.powerdata.Potencia,mysql.aerodata.Data,
mysql.aerodata.Hora
FROM mysql.aerodata,mysql.meteodata,mysql.powerdata
WHERE mysql.aerodata.Hora = mysql.meteodata.HoraM
and mysql.aerodata.Hora = mysql.powerdata.HoraP
GROUP BY mysql.aerodata.Hora
```

Capítulo 5

Resultados de Modelação e Controlo

Neste capítulo foi criado o Modelo real de potência elétrica, o Modelo ideal de potência elétrica e o Modelo de orientação do aerogerador. Estes três modelos foram criados com base nas amostras recolhidas dos sistemas de aquisição de dados.

Tanto o modelo real de potência elétrica como o modelo de orientação do aerogerador foram criados utilizando Redes Neuronais, uma vez que este método permite a identificação de sistemas não lineares. Quanto ao modelo ideal, este foi criado com base nas equações matemáticas que representam a produção da potência elétrica (Fonte eólica). Os modelos de potência elétrica produzida foram comparados de forma a ser estimada a quantidade de potência elétrica perdida.

Como solução para essa perda, foi efetuada a modelação e controlo do travão MR, responsável pela travagem do aerogerador na posição aproximada do vento. Todos os passos intermédios e procedimentos seguidos foram explicados detalhadamente.

5.1 Modelação do Aerogerador - Modelo Real de Potência Elétrica

Para a criação do modelo real de potência elétrica do aerogerador, foram necessárias amostras recolhidas pelos diversos sistemas de aquisição de dados. Estes sistemas foram postos em funcionamento por 5 dias e aproximadamente 6 horas, mais concretamente de 10/02/2017 à 15/02/2017, excedendo o tempo previsto, pois o sistema nem sempre esteve a consumir a corrente máxima utilizada aquando a estimação. Após as medições verificou-se uma tensão elétrica de $3,30V$ nos terminais das baterias, sendo esse valor inferior ao da tensão de operação para o Arduíno Uno, (Arduino, 2017). Algumas das 28072 amostras recolhidas podem ser observadas nas Figuras 5.1 e 5.2.

```
1  SELECT mysql.aerodata.Pitch,mysql.aerodata.Yaw,mysql.meteodata.VelocidadeM,mysql.mete
2  mysql.powerdata.Potencia,mysql.aerodata.Data,mysql.aerodata.Hora from mysql.aerodata,
3  WHERE mysql.aerodata.Data ='2017-02-10' and  mysql.powerdata.DataP ='2017-02-10'
4  and  mysql.meteodata.DataM ='2017-02-10' and mysql.aerodata.Hora = mysql.meteodata.Ho
5  and mysql.aerodata.Hora = mysql.powerdata.HoraP GROUP BY mysql.aerodata.Hora
6
```

SELECT mysql.aerodata.Pit... ×

Page Size: 500000 | Total Rows: 753 Page: 1 of 1 | Matching Rows:

#	Pitch	Yaw	VelocidadeM	PosicaoM	Potencia	Data	Hora
1	3.14	163.27	0.0	103.0	20.082560476403152	2017-02-10	18:02:38
2	1.98	173.92	0.0	105.0	37.50246815976282	2017-02-10	18:02:51
3	3.38	161.89	0.0	123.0	48.192498898498016	2017-02-10	18:02:56
4	2.73	171.79	0.0	129.0	29.626313516956525	2017-02-10	18:03:00
5	3.81	134.39	0.0	139.0	27.550112219248998	2017-02-10	18:03:43
6	3.07	159.29	2.82	113.0	40.03056205075095	2017-02-10	18:03:47
7	1.44	133.57	0.0	129.0	42.11477522766797	2017-02-10	18:04:13

Figura 5.1: Alguns dados adquiridos pelos diferentes sistemas de aquisição de dados e guardados na base de dados.

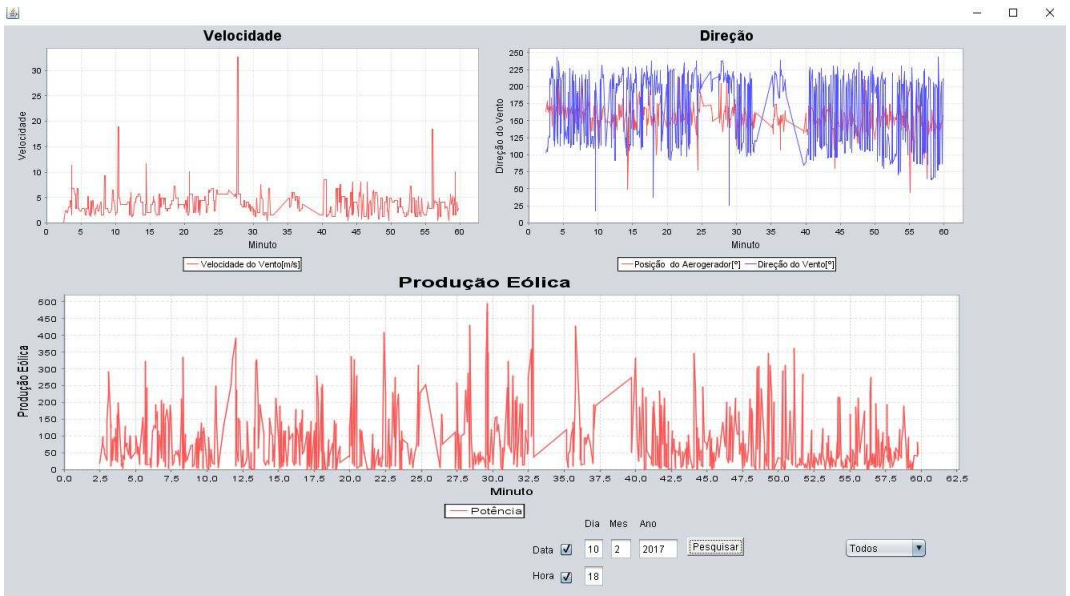


Figura 5.2: Sistema de monitorização - Dados adquiridos as 18 horas de 10/02/2017.

Como se pode observar pela Figura 5.2, existem alguns intervalos de tempo em que não foram recebidos dados. Isto deve-se a possíveis falhas de comunicações originados por fenómenos de reflexão, refração e/ou difração, ou pela falta de sincronismo por pequenos instantes de tempo. Outra justificação possível seria a forma como foi realizada a pesquisa completa na base de dados. A pesquisa selecionava os dados que possuíssem a data e a hora em comum, ou seja, caso existissem dados que não fossem adquiridos na mesma hora, minuto, segundo e data, estes não apareceriam como resultado da mesma.

Quando considerado o tamanho do *buffer* do sensor APC220 conjugado com os possíveis atrasos na comunicação, é possível que aconteça o que é descrito no seguinte exemplo: O sistema de medição do ângulo de *Yaw* do aerogerador enviou dados de 240 em 240ms, sendo este tempo de transmissão reduzido (comparativamente com o tempo de transmissão dos restantes sistemas de aquisição de dados), de forma a garantir que não houvessem perdas significativas de pacotes de dados e sincronismo, precisamente onde não havia fácil acesso. O sistema de aquisição de dados da estação meteorológica enviou os seus dados adquiridos de 1 em 1 segundo, sendo este o tempo de amostragem mínimo associado ao cata-ventos. O sistema de medição da potência elétrica produzida possuía um tempo de aquisição de dados de 4 segundos, devido às limitações das aplicações em funcionamento no PC2.

Estes tempos de amostragem implicam que, quando um pacote de dados chegasse da estação meteorológica, o sensor já teria recebido pelo menos 3 pacotes com a informação relativa ao ângulo de *Yaw* e de *Pitch*. Por sua vez, quando chegasse um pacote com a informação relativa à produção elétrica, o sensor já teria recebido aproximadamente 14 pacotes relativos ao ângulo de *Yaw* e de *Pitch* e, mais 4 pacotes relativos à estação meteorológica. Quando este processo se repete por um longo período de tempo torna possível a perda de pacotes, uma vez que, chegariam pacotes ao recetor no instante em que o seu *buffer* estaria cheio, impossibilitando o armazenamento e o processamento desse pacote. Nas Figuras D.6, D.7 e D.8 pode-se observar pelos campos "Hora", "HoraM" e "HoraP" que os dados não foram recebidos de acordo com o seu tempo de envio, derivado a um dos possíveis acontecimentos descritos.

Foi comparada a direção do vento com a direção do aerogerador, uma vez que se estivessem na mesma direção poderia concluir-se que a produção era 'ótima'. A Figura 5.3 representa as duas direções, mostrando claramente que não estão sempre alinhadas. Este facto reforçou a ideia da criação de um modelo, que permitisse estimar a produção elétrica e concluir sobre as diferenças entre as condições reais e ideias.

Para a criação do modelo real de potência elétrica, foram utilizadas as amostras adquiridas e definidas nove entradas:

- o erro de *Yaw* : (vDir(k-3)-Yaw(k-3)), (vDir(k-2)-Yaw(k-2)) e (vDir(k-1)-Yaw(k-1));
- a velocidade do vento : vVel(k-3), vVel(k-2) e vVel(k-1);
- a potência elétrica : Potencia(k-3), Potencia(k-2) e Potencia(k-1).

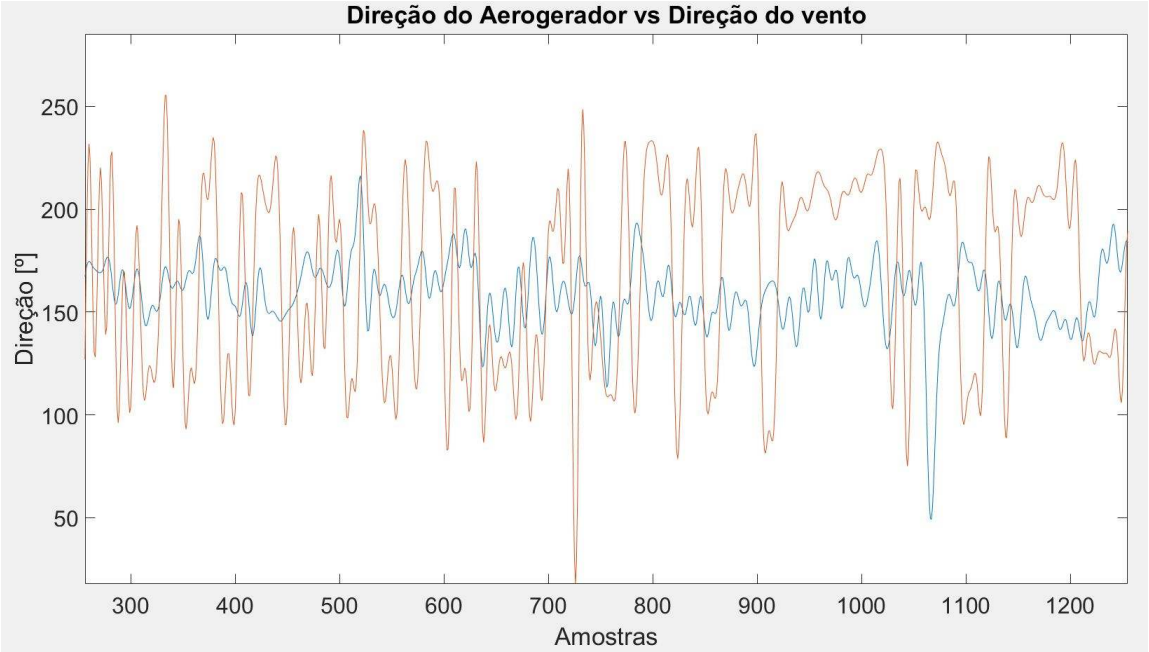


Figura 5.3: Direção do vento vs Direção do aerogerador, Ta = 0,8 segundos.
Direção do vento Direção do aerogerador

Quanto a saída esta foi definida como sendo a potência elétrica no instante (k), $Potencia(k)$. Tendo em conta as mudanças quase instantâneas nas amostras, foi realizada uma sobre-amostragem, de forma a acrescentar cinco vezes mais amostras e atenuar essas mudanças quase instantâneas, provocadas em grande parte por perdas de pacotes transmitidos durante a aquisição de dados e pela pesquisa realizada à base de dados (Problema discutido anteriormente). Após a sobre-amostragem obteve-se 140360 amostras e estas foram filtradas utilizando um Filtro de Kalman, com o objetivo de eliminar as mais altas frequências. Nas Figuras 5.4, 5.5, 5.6 e 5.7, estão representadas aproximadamente 1200 das 140360 amostras. É de frisar que não há grandes diferenças entre os sinais filtrados e os originais, uma vez que sendo estes sistemas não lineares, algumas das altas frequências são importantes para a criação de um bom modelo, Anexo D.

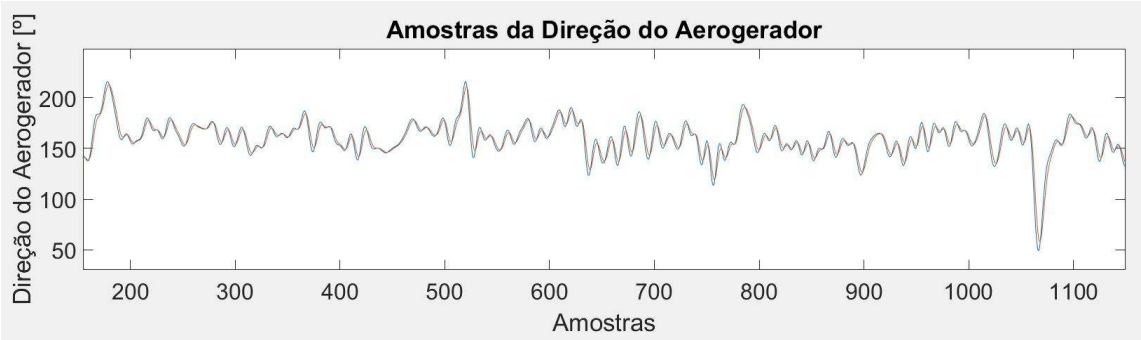


Figura 5.4: Dados do ângulo de *Yaw* do aerogerador e a aplicação do Filtro de Kalman com os coeficientes [$Q = 3$ e $R = 10$], Ta = 0,8 segundos.
Direção do aerogerador Direção do aerogerador filtrado

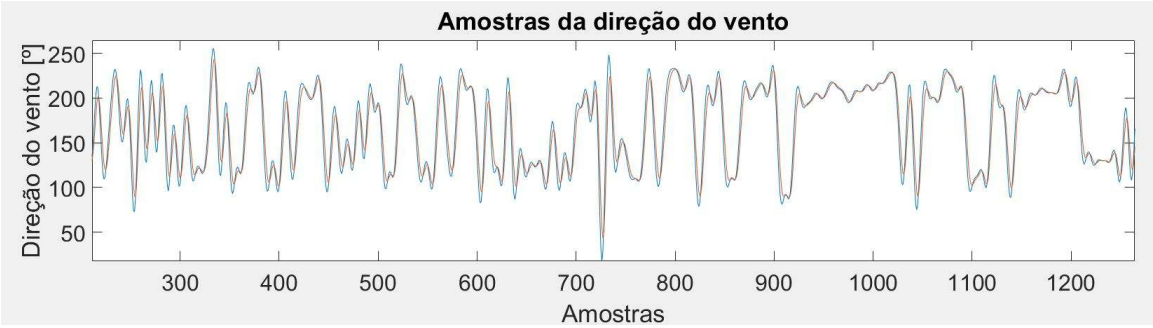


Figura 5.5: Dados da direção do vento e a aplicação do Filtro de Kalman com os coeficientes [$Q = 3$ e $R = 10$], Ta = 0,8 segundos.

Direção do vento Direção do vento filtrado

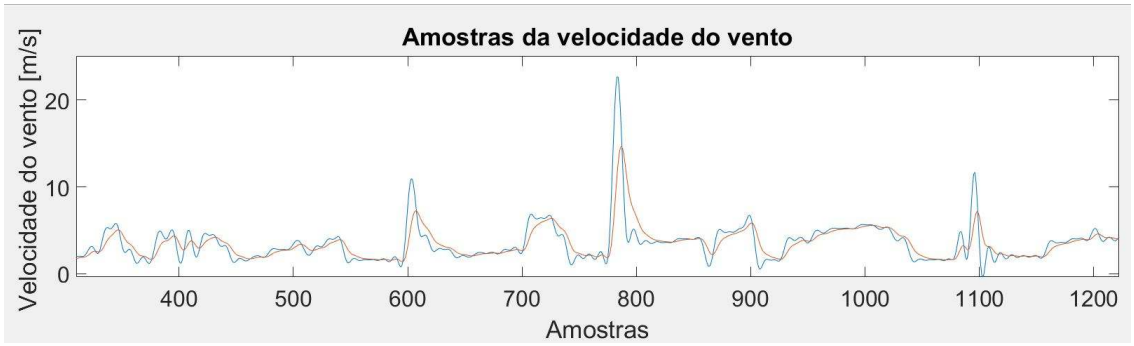


Figura 5.6: Dados da velocidade do vento e a aplicação do Filtro de Kalman com os coeficientes [$Q = 0,1$ e $R = 5$], Ta = 0,8 segundos.

Velocidade do vento Velocidade do vento filtrado

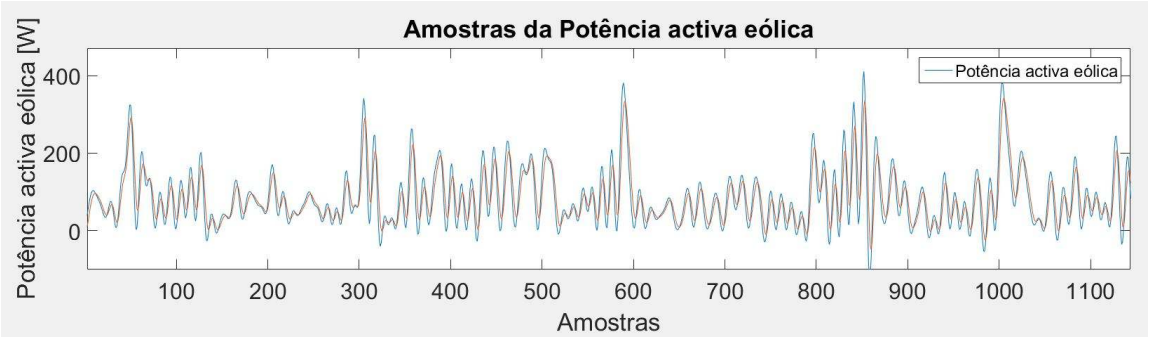


Figura 5.7: Dados da produção eólica e a aplicação do Filtro de Kalman com os coeficientes [$Q = 10$ e $R = 50$], Ta = 0,8 segundos.

Potência elétrica produzida Potência elétrica filtrado

Utilizando a ferramenta *nftool* do *Matlab R2015b*, foram definidos 9 neurónios internos e um neurónio externo. A rede foi treinada com 70% das amostras por 1000 épocas, utilizando o algoritmo *Scaled Conjugate Gradient*, uma vez que se pretendia um modelo generalista. As restantes 30% das amostras foram divididas e utilizadas para validação e teste.

Após o treino foram obtidos os seguintes resultados, relativamente ao erro médio quadrado (MSE) e o fator de correlação (R), Figura 5.8. Mesmo quando simulado com 100% das amostras, o resultado da simulação foi satisfatório, como pode ser verificado pela Figura 5.9. Relativamente ao formato da rede, este está representado na Figura 5.10 e, um resumo dos parâmetros utilizados para a implementação deste modelo pode ser observado na Tabela 5.1.

Results

	Samples	MSE	R
Training:	126324	7.30021e-5	9.96600e-1
Validation:	7018	7.41950e-5	9.96540e-1
Testing:	7018	7.15623e-5	9.96524e-1

Figura 5.8: Erro médio quadrado e fator de correlação.

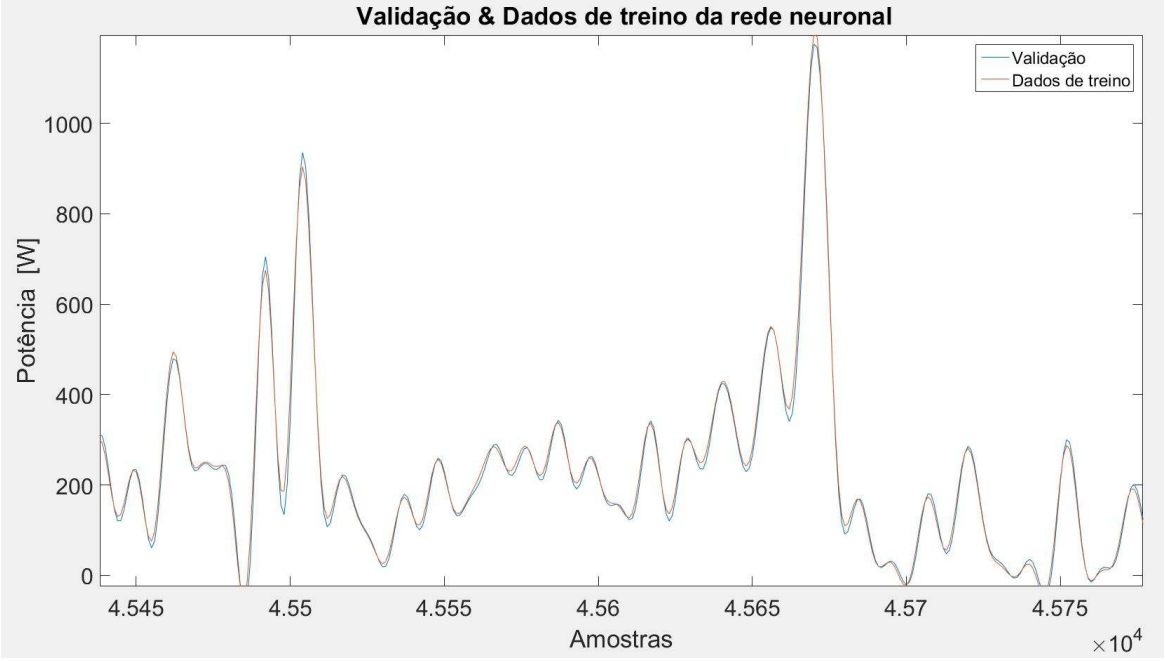


Figura 5.9: Validação do Modelo Real de Potência Elétrica produzida no intervalo de amostras [45000;458000] .

Validação Dados de treino

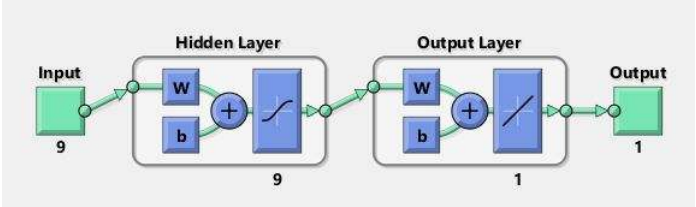


Figura 5.10: Formato da rede neuronal, modelo real da produção eólica.

Tabela 5.1: Parâmetros utilizados na implementação do do Modelo Real de Potência Elétrica.

Parâmetros/ Dados adquiridos	**Ângulo de Yaw**	**Direção do vento**	**Velocidade do vento**	**Potência elétrica Produzida**
Tempo de aquisição de dados	5 dias e 6 horas			
Data de aquisição	10/02/2017 - 15/02/2017			
Total de amostras adquiridas	28072			
Total de amostras com sobre-amostragem	140360			
Filtro de Kalman - Parâmetro Q	3	3	0,1	10
Filtro de Kalman - Parâmetro R	10	10	5	50

5.2 Modelação do Aerogerador - Modelo Ideal de Potência Elétrica

Tal como descrito na Secção 3.7 do Capítulo 3, para a criação do modelo ideal era necessário calcular a potência do vento e a potência mecânica retirada do vento. Não conhecendo a potência mecânica, esta poderia ser estimada com base na potência elétrica real produzida, desde que fosse utilizado um rendimento elétrico.

À semelhança do trabalho realizado em (Afonso, 2010), e, considerando a redução do rendimento com o tempo, foi admitido um rendimento elétrico de $\eta = 85\%$. Quanto ao cálculo da densidade da massa de ar ρ, não se possuía dados para o seu cálculo exato, logo a solução foi utilizar o valor padrão de $\rho = 1,225$ para uma temperatura de $15°C$ e pressão do ar de $1atm$. De forma a verificar se existiam condições aproximadas para aplicação do valor padrão, foram considerados os valores máximos e mínimos de temperatura durante o período de aquisição de dados. A mesma consideração foi feita quanto à altitude.

No que diz respeito à pressão do ar e considerando os dados relativos à localização da Faculdade de Ciências e Tecnologias, situada a uma altitude comprendida entre $100m$ e $125m$ em relação ao nível do mar, (Almada Informa, 2017), esta possui nos limites inferior e superior uma pressão do ar de $P_r(H_{TP}) = 0,9882atm$ e $P_r(H_{TP}) = 0,9853atm$. Relativamente à temperatura, esta atingiu a sua máxima no dia 15 de Fevereiro de 2017 com aproximadamente $18°C$ e a sua mínima foi no dia 10 do mesmo mês com $5°C$, (Tempo, 2017). No entanto, não se conseguiu precisar a que altitude foi realizada a medição da temperatura, logo o ajuste não pôde ser considerado como exato. Por este motivo assumiu-se que a temperatura tinha sido os $15°C$ necessários para a aplicação do valor padrão.

Assumindo as condições padrão passou-se à fase da escolha do valor do coeficiente de potência C_p, que maximizasse a potência mecânica retirada do vento. Uma vez que o

modelo de produção ideal é calculado de forma teórica, faz sentido que o C_p também o seja. Com base neste raciocínio foi utilizado o C_p = 40%, tendo sido este o valor máximo observado por (Afonso, 2010).

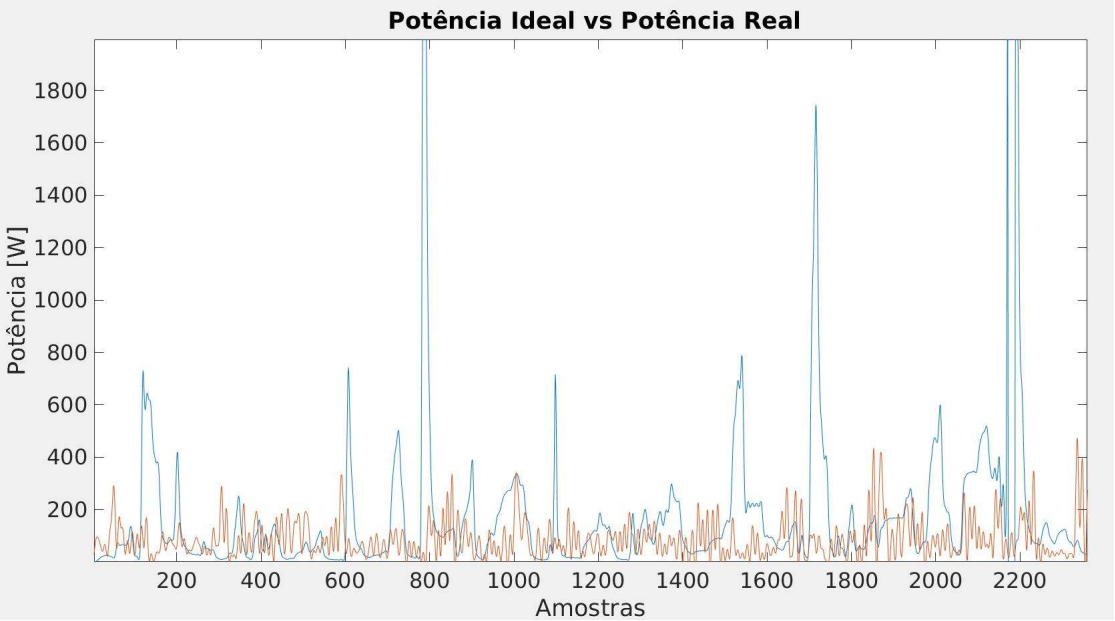


Figura 5.11: Potência ideal estimada vs Potência real adquirida, Ta = 0,8 segundos.
Potência elétrica ideal (Estimada) Potência elétrica real (Adquirida)

Pela Figura 5.11, observou-se que haviam diferenças entre a produção real e ideal. Os picos que aparecem na produção devem-se às rajadas de vento, uma vez que na Equação 3.11, a velocidade do vento está ao cubo. Por outro lado, a velocidade máxima do vento de segurança para este aerogerador é de $16m/s$, o que implica que quando o vento atinge este valor o aerogerador deixa de produzir. Esta característica foi implementada no modelo, forçando a produção a ser nula sempre que a velocidade do vento fosse superior à $16m/s$. No entanto, quando observada a produção real adquirida verificou-se que esta nem sempre foi nula para valores elevados de velocidade do vento. Um dos motivos para este acontecimento é o facto deste aerogerador dificilmente estar alinhando com a direção do vento, uma vez que o seu sistema de seguimento da direção do vento possui uma inclinação, Figura 5.12, não permitindo que toda a área das pás do aerogerador esteja em contacto com o vento.

Observa-se ainda na Figura 5.13 que a potência perdida era significativa, comparativamente à potência elétrica média produzida, então, um sistema de controlo da direção foi implementado de forma a se reduzir as diferenças de produção. As figuras relativas aos modelo real e ideal de potência elétrica, assim como a potência elétrica perdida, estão disponíveis no Anexo E.

Figura 5.12: Aerogerador e o desalinhamento existente entre a cauda e a turbina.

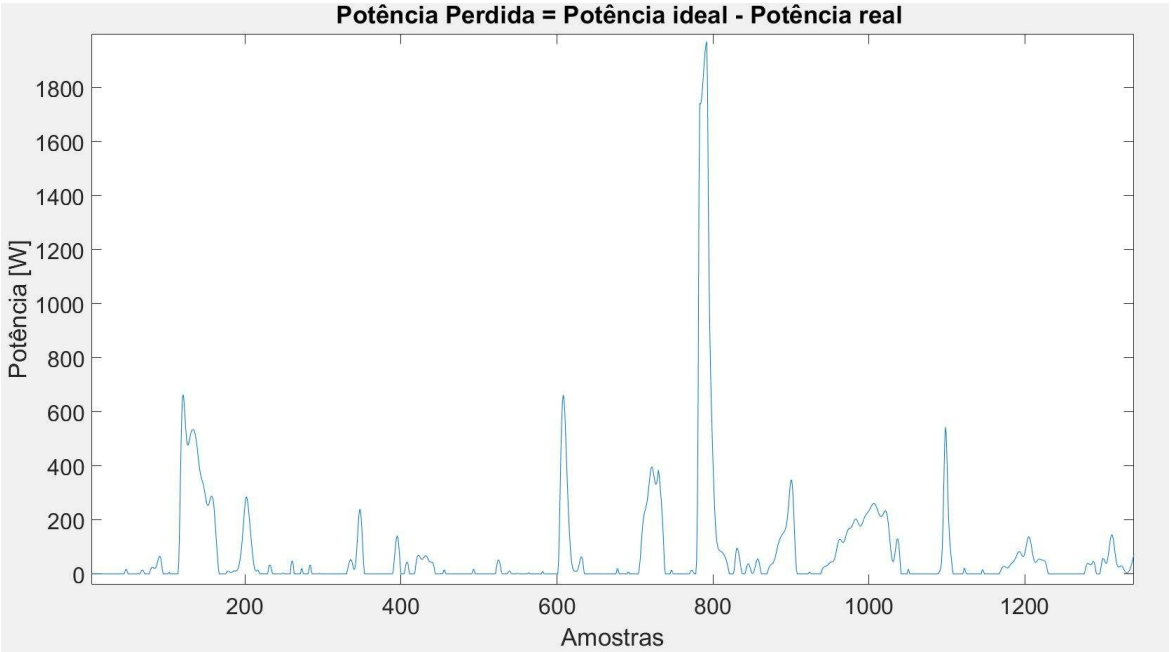


Figura 5.13: Potência perdida quando considerada uma produção ideal, Ta = 0,8 segundos.
Potência perdida

5.3 Implementação do Modelo de Bingham

Com o objetivo de implementar o controlador referido na secção anterior, o Modelo de Bingham foi testado de forma a se observar o seu comportamento, utilizando as bases explicadas na Secção 3.8.2 do Capítulo 3 . Este teste foi realizado, considerando um sistema passivo, onde os coeficientes da equação da força aplicada por este modelo possuíam um valor constante. Foi implementada a representação do modelo de Bingham em Espaços de Estados, utilizando valores aceitáveis (aleatórios e dentro da gama dos valores encontrados noutros trabalhos) de coeficiente de amortecimento, coeficiente de suspensão e massa associada ao objeto amortecido, (Florin e Liliana, 2013), tendo sido observada a sua resposta ao impulso.

Sendo a força F dada pela Equação 5.1,

$$F = m\ddot{x} = -f_c - c_0\dot{x} + U_a \tag{5.1}$$

em que U_a é a perturbação introduzida e f_c é dada pela Equação 5.2, onde k_s é o coeficiente de elasticidade. Recorrendo à substituição (Equação 5.3) e simplificação em função de $\ddot{x}$, a equação original da força aplicada torna-se na Equação 5.4.

$$f_c = k_s x \tag{5.2}$$

$$m\ddot{x} = -k_s x - c_0\dot{x} + U_a \tag{5.3}$$

$$\ddot{x} = \frac{-k_s x}{m} - \frac{c_0\dot{x}}{m} + \frac{U_a}{m} \tag{5.4}$$

Considerando x_1 e x_2 as variáveis de estado com as igualdades descritas pelas Equações 5.5 e 5.6, são obtidas as relações descritas pelas Equações 5.7 e 5.8.

$$x_1 = x \tag{5.5}$$

$$x_2 = \dot{x} \tag{5.6}$$

$$\dot{x_1} = \dot{x} = x_2 \tag{5.7}$$

$$\dot{x_2} = \ddot{x} \tag{5.8}$$

Substituindo as variáveis de estado na Equação 5.4 obtém-se a Equação 5.9,

$$\dot{x_2} = \frac{-k_s x_1}{m} - \frac{c_0 x_2}{m} + \frac{U_a}{m} \tag{5.9}$$

Obtida uma igualdade para cada um dos estados, $\dot{x_1}$ e $\dot{x_2}$, a representação do modelo de Bingham em Espaço de Estados é dada pelas Equações 5.10 e 5.11.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{-k_s}{m} & \frac{-c_0}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} U_a \tag{5.10}$$

$$Y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} U_a \tag{5.11}$$

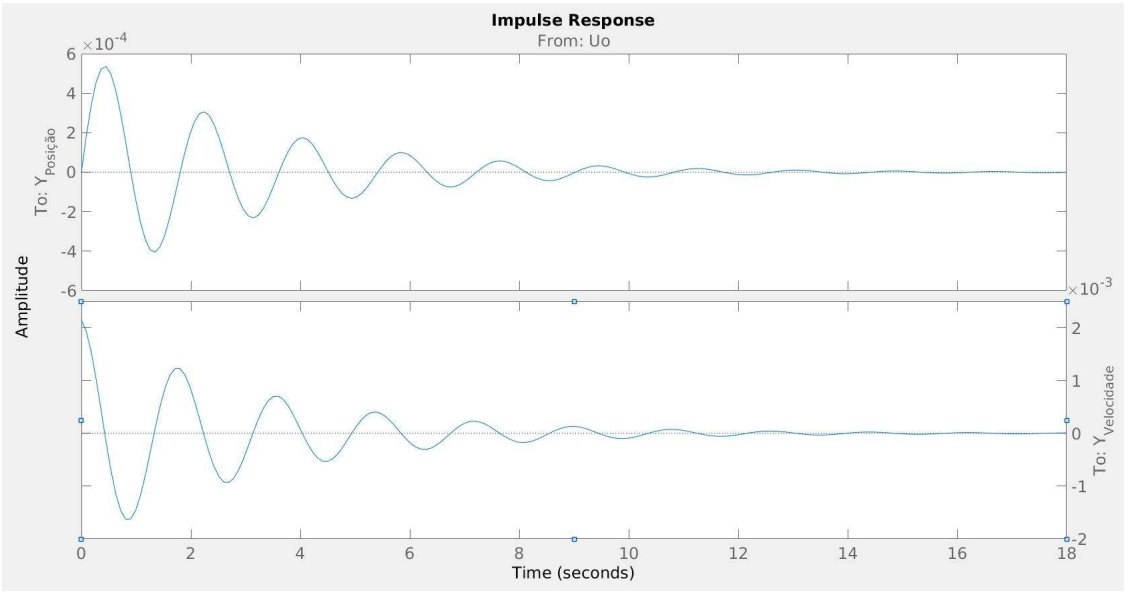


Figura 5.14: Resposta ao impulso do modelo de amortecedor de Bingham, a funcionar como amortecedor passivo .

A Figura 5.14 representa a resposta ao impulso, simulado com

$k_s = 5700N/m$;

$m = 466,5kg$;

$c_0 = 290Ns/m$.

Pela resposta ao impulso é percetível o comportamento de um amortecedor representado pelo modelo de Bingham.

5.4 Implementação do Binário de Travagem

Após o teste do modelo de Bingham, foram identificados os coeficientes e as variáveis necessárias para a implementação e, a simulação do modelo de Bingham em função do seu binário de travagem. Como observado na Secção 5.3, o modelo de Bingham possui o comportamento de um amortecedor, quando considerados coeficientes fixos num ambiente com movimento linear. Como explicado na Secção 3.9 do Capítulo 3, este deve ser implementado em função do binário de travagem, uma vez que se pretende travar um objeto com movimento rotacional.

Tendo sido escolhido o fluído MR (*MRF-140CG Magneto-Rheological Fluid*) e com base nos dados disponibilizados no seu *Datasheet* (Figuras 3.23, 3.24 e 3.25), o comportamento do fluído foi simulado. Utilizando a ferramenta *Cftool* do *Matlab R2015b* foram calculados os coeficientes K_i, que formam um polinómio de terceira ordem com um comportamento

aproximado da curva da tensão de escoamento da Figura 3.23. Para tal foi necessário retirar valores da tensão de escoamento e o respetivo campo magnético aplicado, sendo os valores utilizados expostos em matrizes no *MATLAB*,

$H = [\ 40\ 80\ 120\ 160\ 200]$ $\qquad$ $YieldStress = [20,5\ 37,5\ 48\ 54,8\ 59]$

e os coeficientes K_i calculados são representados, Figura 5.15.

$K_1 = 0,7414$ $\qquad$ $K_2 = -0,0031$ $\qquad$ $K_3 = 5,0780^{-6}$

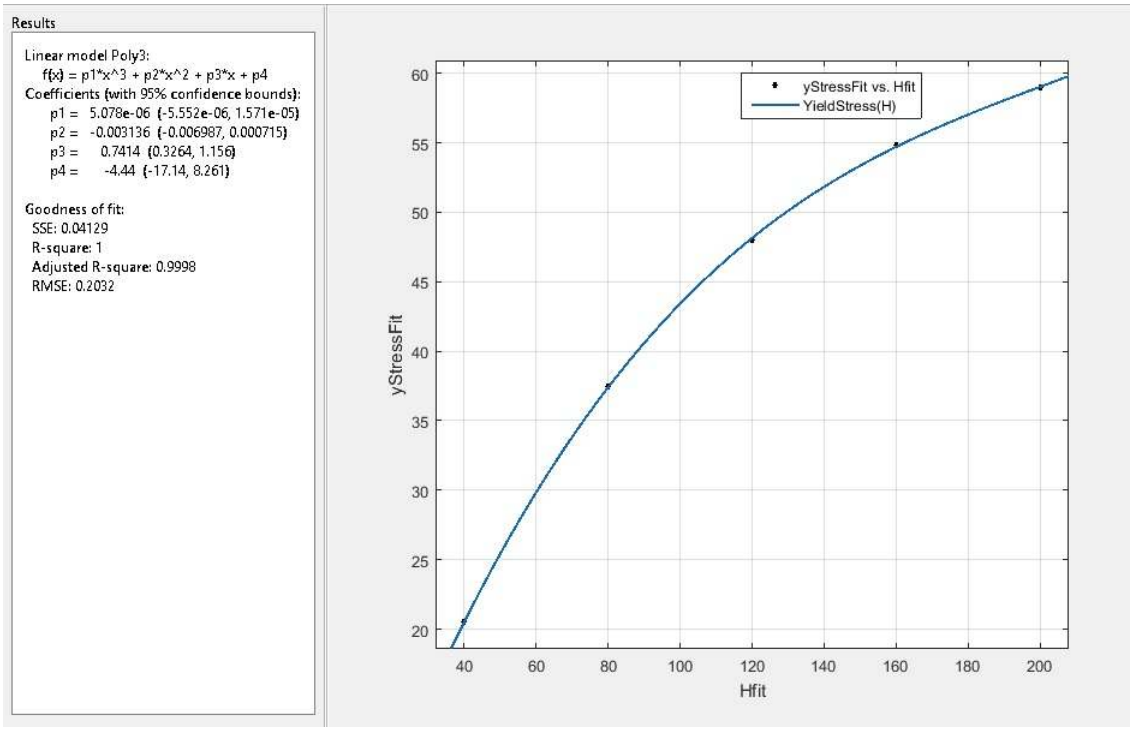


Figura 5.15: Aproximação da curva da tensão de escoamento - Matlab "cftool".

Com estes coeficientes foi obtida a Equação 5.12 que representa o tensão de escoamento em função do campo magnético.

$$\tau_y = 0,7414B - 0,0031B^2 + 5,0780^{-6}B^3 \tag{5.12}$$

Com base na Equação da tensão de corte, Equação 3.18, conclui-se que a Equação 3.25 é equivalente a Equação 5.13, descrevendo o binário de travagem de forma a que o único elemento a determinar na equação seja a viscosidade ν, visto que $\dot{\gamma}$ é dada pela Equação 5.14, (Poznić et al., 2012).

$$T_{MR} = \pi N(\frac{4}{3}(\tau_y + \nu\dot{\gamma})R_o^3 + \nu\frac{\omega}{g}R_0^4) + T_{fric} \tag{5.13}$$

$$\dot{\gamma} = \frac{R_o \omega}{g} \tag{5.14}$$

Quanto à viscosidade, esta foi calculada do mesmo modo que a tensão de escoamento, ou seja, calculando os coeficientes de um polinómio que representasse a curva da Figura 3.25, Figura 5.16. Foi determinado o declive da reta de segundo a segundo de modo a que este representasse a viscosidade. Para o presente estudo foi escolhido o menor valor de viscosidade, de forma a ser considerada a menor resistência ao movimento, Figura 5.17, $\nu = 0,2404$.

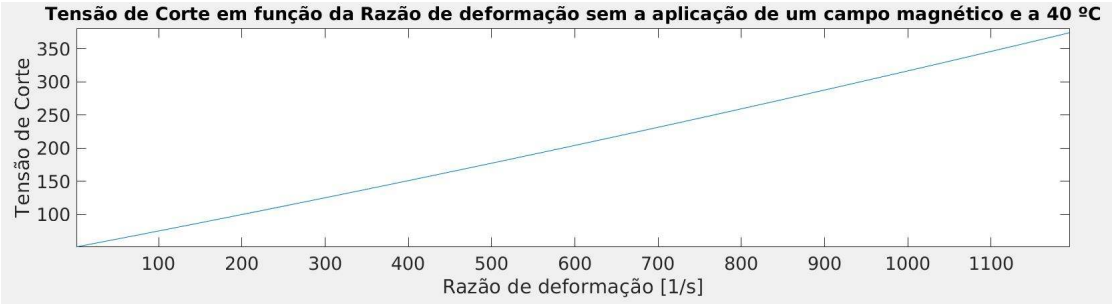


Figura 5.16: *Matlab*- Tensão de corte em função da Razão de deformação sem a aplicação de um campo magnético a 40ºC.

Tensão de corte (Razão de deformação)

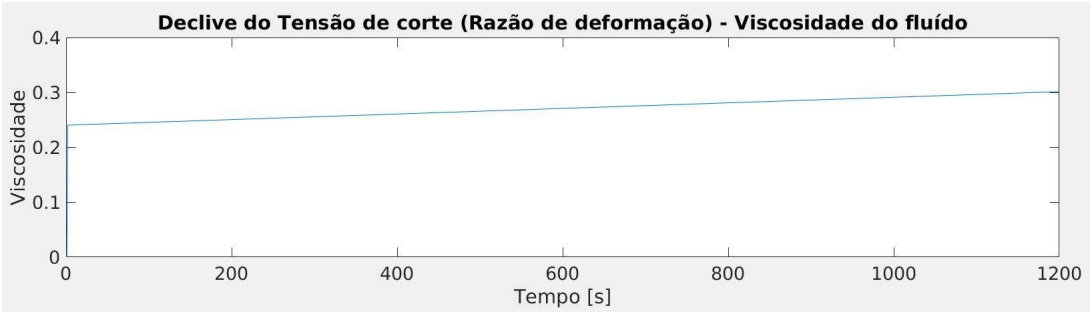


Figura 5.17: *Matlab*- Declive da tensão de corte em função da Razão de deformação, Viscosidade do fluído.

Viscosidade(t)

Na Figura 5.18 está representado o diagrama de blocos criado no *Matlab/Simulink 2015b*, do simulador da tensão de corte τ do fluído MR em função do campo magnético aplicado. O mesmo foi inserido no modelo matemático de binário escolhido, tal como representado na Equação 5.13, sendo a entrada do modelo uma corrente elétrica gerada a partir do controlador difuso de binário.

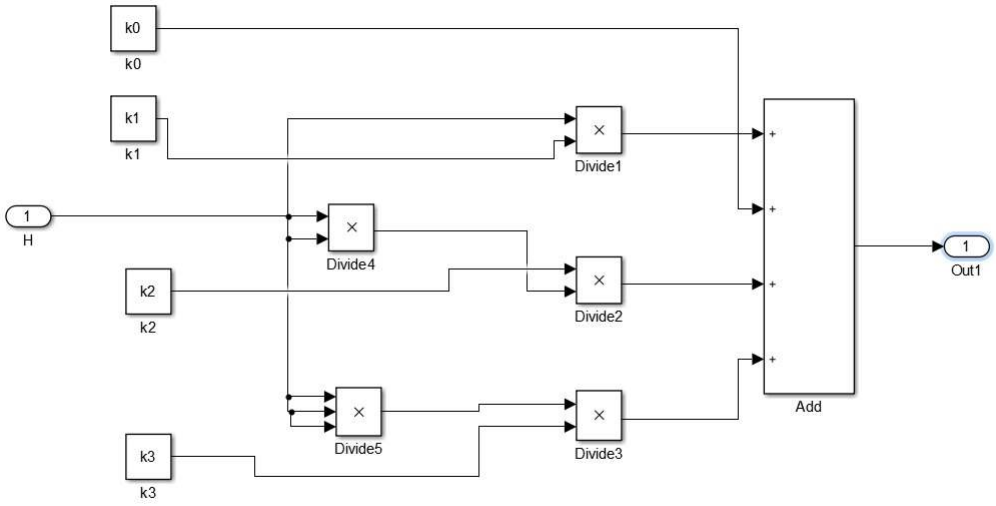


Figura 5.18: Matlab/Simulink- Diagrama de blocos do simulador da tensão de escoamento.

Esta corrente elétrica foi controlada de forma a possuir tanto valores positivos como negativos, representando assim uma inversão no sentido da mesma, e, como resultado, o modelo de travão escolhido gerava binários com o sentido da corrente elétrica. Este facto implicava que o modelo de travão não apenas serveria para travar (subtrair binário ao aerogerador) mas também para acelerar (somar binário) o aerogerador, o que na prática se torna incompreensível. Um travão deveria apenas travar, com maior ou com menor intensidade, mas nunca acelerar. Mesmo considerando que esta seria uma limitação do modelo de travão escolhido, este foi implementado por se revelar uma ideia interessante (uma configuração que permitisse travar e acelerar, dependendo apenas do sentido da corrente elétrica) e o seu binário foi controlado tal como será explicado posteriormente.

5.4.1 Controlador de Binário

Para a implementação do controlador de binário foi utilizado o controlador difuso com a inferência de Mamdami. Este controlador possui uma entrada e uma saída, sendo a entrada o erro de binário e a saída uma corrente elétrica que funcionaria como ação de controlo (Capítulo 3 - Secção 3.11), Figura F.1. Como entrada do controlador, foram utilizadas funções de pertença do tipo "gbellmf ", no intervalo $[-20; 20] N.m$, em que as três funções de pertença representavam o binário negativo, o binário quase nulo e o binário positivo, com os seguintes parâmetros:

- Binário negativo : [15,46 3,278 -16,03];
- Binário quase nulo : [0,5228 2,5 0];
- Binário positivo : [16,45 3,278 17,01].

Quanto à saída do controlador, as funções de pertença eram do tipo "gauss2mf" e geravam correntes elétricas no intervalo $[-2;2]A$. As três funções de pertença representavam a corrente elétrica negativa, a corrente elétrica quase nula e a corrente elétrica positiva, com os seguintes parâmetros:

- Corrente negativa : [0,003398 -2 0,1672 -0,444];
- Corrente quase nula : [0,1359 -0,09 0,1359 0,09];
- Corrente positiva : [0,1672 0,4392 0,003398 2].

Para terminar, foi utilizado o método de desdifusificação "Centroid" e foram definidas as seguintes regras :

```
1 Se (Binário <0) então (Corrente <0)
2
3 Se (Binário >0) então (Corrente >0)
4
5 Se (Binário > -0,5228 & Binário < 0,5228) então
6  (Corrente > -0,09 & Corrente < 0,09)
```

No teste do sistema com o controlador, foi necessário definir valores da massa do objeto em rotação m, o valor de g, do raio exterior do disco do travão MR R_o, do raio da espira R_{espira}, do número de superfícies do disco em contacto com o fluído N e do raio do eixo de rotação do aerogerador R.

Estes parâmetros físicos foram escolhidos tendo como base valores típicos encontrados nos estudos realizados, de forma a se manterem o mais próximos da realidade,

$R_{espira} = 1^{-3}$ $R_o = 200 \times 10^{-3}$ $g = 10 \times 10^{-3}$ $R = 100 \times 10^{-3}$ $N = 1$
$m = 60$

sendo R_{espira}, R_o, R e g apresentados em metros, m em Kg e ν em $[Pa.s]$.

Após a simulação do diagrama de blocos do controlador de binário, foi possível observar que o binário total do sistema seguia aproximadamente o comportamento do binário de referência (Figura 5.19), sendo a gestão da corrente elétrica (Figura 5.20) realizada pelo mesmo controlador.

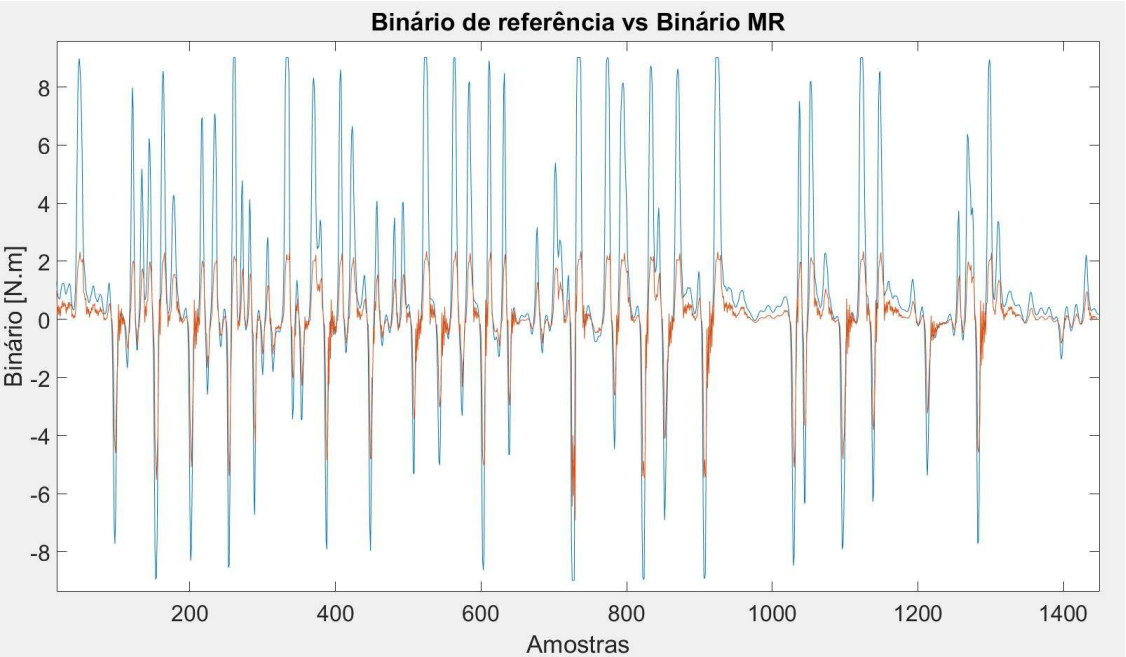


Figura 5.19: Matlab/Simulink- Binário total controlado por corrente elétrica, Ta = 0,8 segundos.

Binário de referência Binário total do sistema

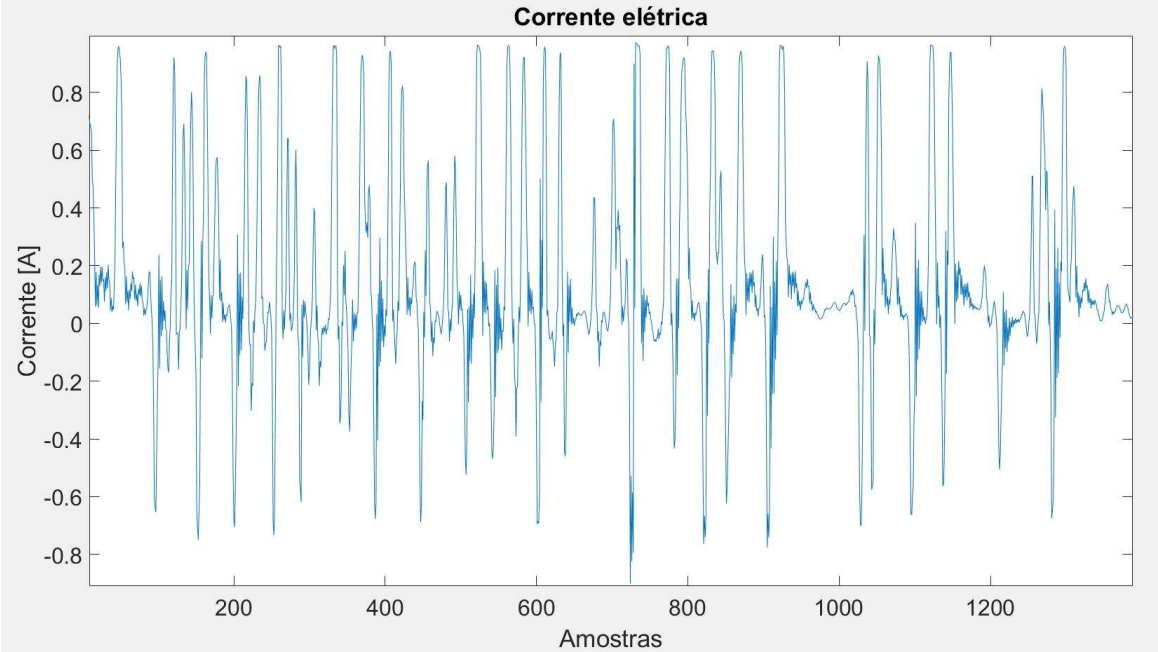


Figura 5.20: Matlab/Simulink- Corrente elétrica produzida pelo controlador difuso, Ta = 0,8 segundos.

Corrente elétrica

5.4.2 Controlo de posição

Para implementar o controlo de posição foram necessários três novos blocos no *Simulink*:

- o modelo que representa o movimento de rotação do aerogerador face às variações do erro de *Yaw*, a velocidade do vento e a aceleração do aerogerador;
- o bloco de saturação e acerto da posição angular;
- controlador de posição.

O modelo foi gerado com base nas redes neuronais, tendo 12 neurónios na camada interna, 1 na camada externa e, para a regressão foram consideradas amostras até "k-3" (Figura 5.21), da seguinte forma:

```
1
2       uum(k,:) =  [accelAero(k-3) accelAero(k-2) accelAero(k-1) ...
3                    vDir(k-3) vDir(k-2) vDir(k-1) ...
4                    vVel(k-3) vVel(k-2) vVel(k-1) ...
5                      Yaw(k-3)  Yaw(k-2)  Yaw(k-1)];
6
7                yym(k,:) = [yaw(k)];
```

uum representa a entrada do modelo, *accelAero* é a aceleração angular do eixo de rotação do aerogerador (calculada com base no "Yaw") e *yym* representa a saída do modelo.

Foi utilizado o algoritmo de "Levenberg-Marquardt" com o objetivo de atingir um erro mínimo de 1×10^{-6}. Como resultado obteve-se uma boa aproximação da realidade, Figura 5.22, com $MSE = 9,8196e-07$.

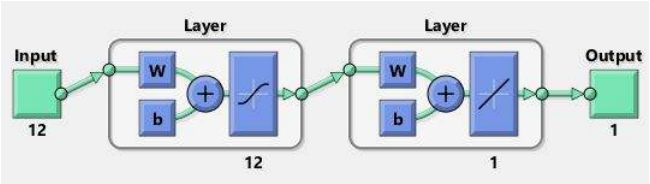


Figura 5.21: Formato da rede neuronal, modelo de direção do aerogerador.

Quanto ao bloco de saturação, este deveria limitar as posições angulares entre $-360°$ e $360°$, de forma a ser considerado o sentido da rotação. O controlador de binário apenas sabe como reagir nesse intervalo de valores, o que implica que, caso o erro de posição fosse inferior a $-360°$ ou superior a $360°$, o controlador não saberia como reagir e criaria ações de controlo fora dos limites.

A Figura 5.23 representa os blocos de saturação e acerto dos deslocamentos angulares negativos e positivos, ou seja, considerando o sentido da rotação.

Quanto ao controlador de posição, este foi implementado com base no controlo difuso, tal como o controlador de binário, tendo sido utilizada a inferência de Mamdami.

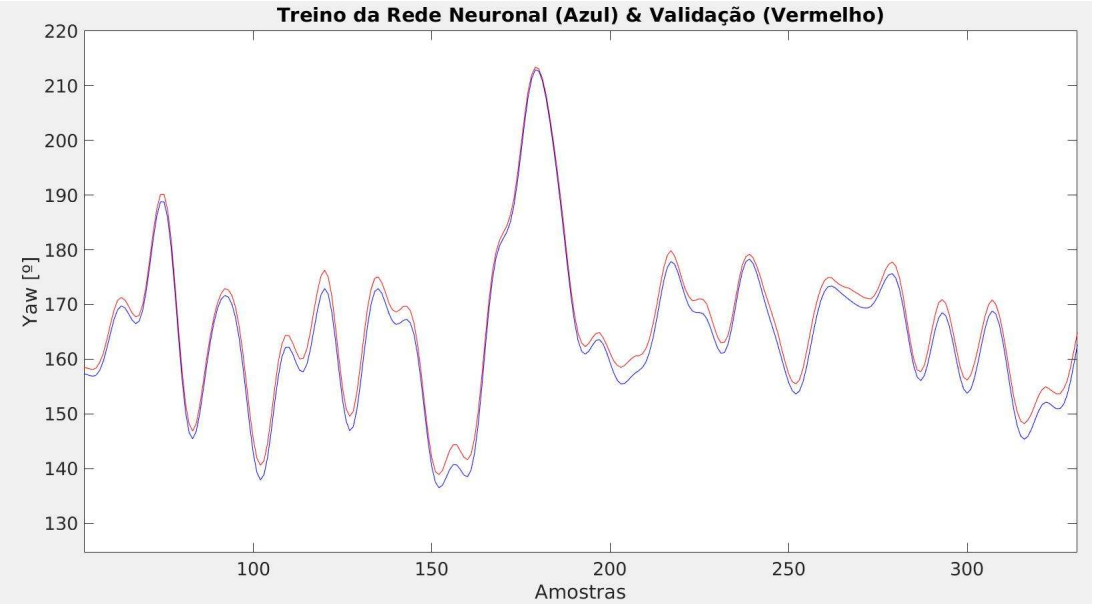


Figura 5.22: Modelo de direção do aerogerador.
Dados de treino Validação

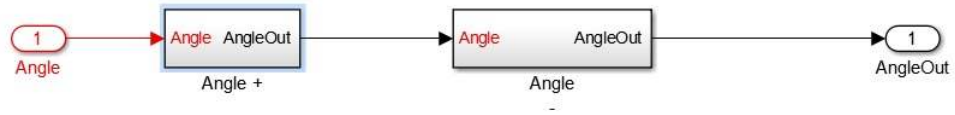


Figura 5.23: Matlab/Simulink- Bloco de correção da posição.

Também possui uma entrada e uma saída, sendo a entrada o erro de posição e a saída um binário de referência (Capítulo 3 - Secção 3.11). Para a criação do controlador de posição, foram utilizadas três funções de pertença do tipo "gauss2mf" à entrada do controlador, no intervalo $[-360;360]$, representando a orientação no sentido negativo, a orientação no sentido positivo e o erro próximo de 0°, com os seguintes parâmetros, Figura F.2:

- Sentido negativo : [-81 -360 -30,58 -81];
- 0° : [3,885 -2,923 3,885 2,923];
- Sentido positivo : [30,58 81 110,5 360].

Na saída, as funções de pertença eram do tipo "gbellmf" e trabalhavam no intervalo de $[-20;20]Nm$. As três funções de pertença representavam o binário negativo, o binário quase nulo e o binário positivo, com os seguintes parâmetros:

- Binário negativo: [14,13 3,278 -16,03];
- Binário quase nulo : [1,823 4,097 0];
- Binário positivo : [15,11 3,278 17,01].

Para terminar o dimensionamento do controlador de posição, tal como no controlador de binário, foi utilizado o método de desdifusificação "Centroid" e foram definidas as seguintes regras :

```
1 Se (Erro_de_Posição ~< 0) então (Binário ~<0)
2
3 Se (Erro_de_Posição ~> 0) então (Binário ~>0)
4
5 Se (Erro_de_Posição > -2,923 & Erro_de_Posição <  2,923) então
6  (Binário > -1,823 & Binário <1,823)
```

Por fim, o controlador de posição, o modelo de orientação do aerogerador e os blocos de saturação foram introduzidos no anel de controlo e este foi simulado. Após a simulação, observou-se que o aerogerador seguia melhor a direção do vento com o controlador, Figura 5.24, com um erro de *Yaw* médio de 14°, o que é inferior ao erro médio de 22,7° sem a aplicação do controlador. O modelo de Bingham, aqui aplicado, é um modelo de travão MR, no entanto proporcionou também um seguimento da direção do vento quando aplicado o controlador. Este controlador, como se pode observar pelas regras impostas, não apenas emite uma ação de controlo quando o aerogerador atinge o intervalo de erro de posição de $[-2,923; 2,923]°$, o que deveria ser feito numa situação de travagem. Este, também reage fora desse intervalo (regras para as condições *Erro_de_Posição* < 0 e *Erro_de_Posição* > 0), o que implica que o binário é controlado constantemente de forma a proporcionar um seguimento da referência (direção do vento). Realizando o seguimento da direção do vento, o erro de *Yaw* foi reduzido, Figura 5.25, o que pela teoria deveria aumentar ligeiramente a potência elétrica produzida.

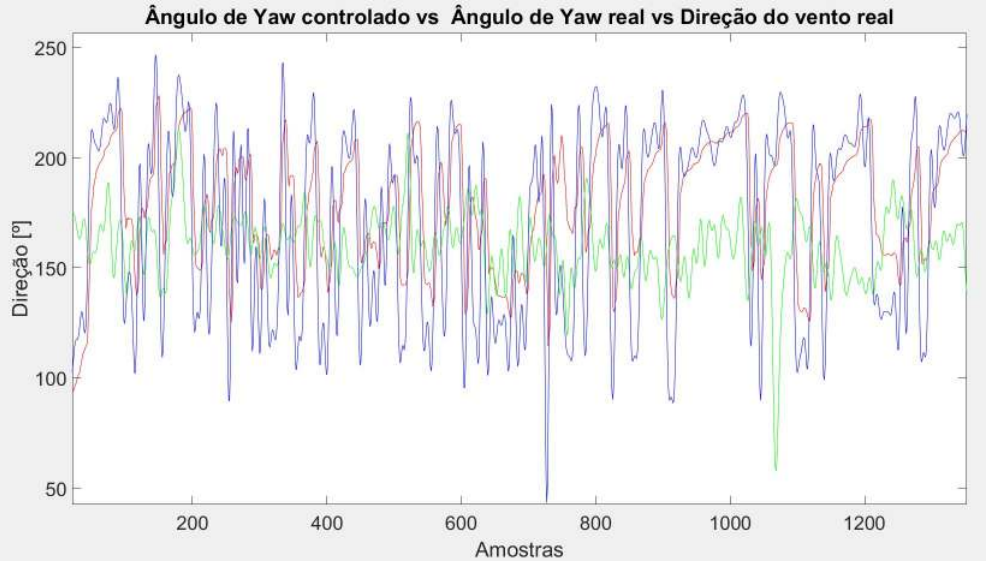


Figura 5.24: Matlab/Simulink- Direção controlada do aerogerador vs Direção real do aerogerador vs Direção do vento, Ta = 0,8 segundos.

Ângulo de yaw controlado Direção do vento Ângulo de yaw real

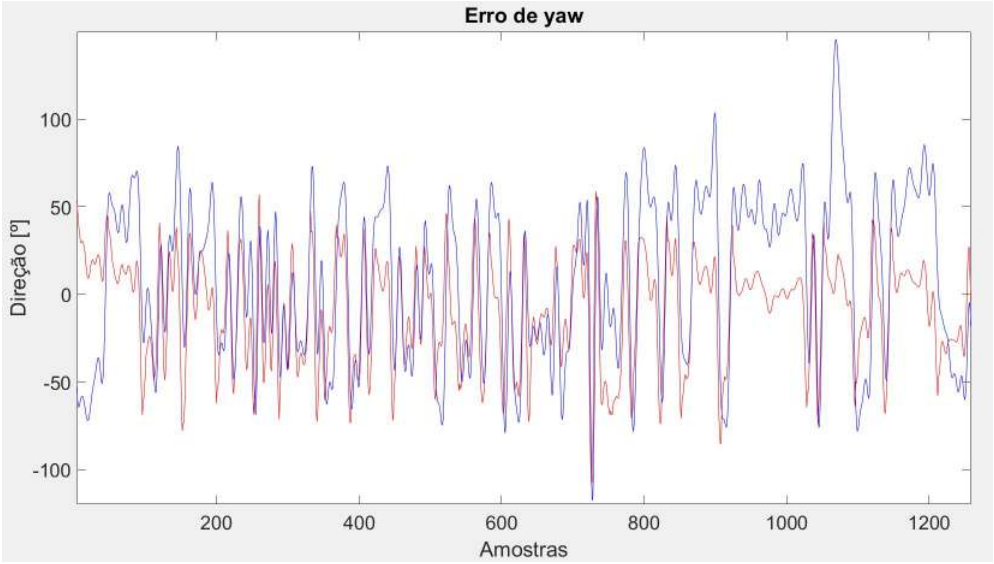


Figura 5.25: Matlab/Simulink- Erro de alinhamento (Erro de *yaw*) com a direção do aerogerador controlada vs Erro de alinhamento com a direção real do aerogerador, Ta = 0,8 segundos.

Erro de *Yaw* controlado Erro de *Yaw* real

5.5 Discussão de Resultados

Nesta secção, são apresentados e discutidos os resultados obtidos com e sem a aplicação do controlo. Os modelos gerados com base nas redes neuronais foram testados nas diferentes condições do vento: quase estacionário, rajadas de vento e mudanças rápidas da direção do vento. Do ponto de vista da produção, esta foi analisada e comparada com controlo, sem controlo e em condições ideais. Para a comparação dos resultados, foram retiradas 8000 amostras dos intervalos que mais se adaptavam às condições descritas.

5.5.1 Simulação - Vento quase estacionário

Na condição de vento quase estacionário foram escolhidas amostras no intervalo [100000;108000] (Figura 5.26) no qual não haviam grandes variações na direção e na intensidade do vento. Utilizando o filtro de Kalman com os parâmetros $R = 20$ e $Q = 0,001$, as amostras foram filtradas de forma a tornar a direção do vento mais estacionária. Nesta condição foi observada a produção real no intervalo de amostras escolhido, Figura 5.27, e estimada a produção ideal, Figura H.1. A produção real foi em média $14,8W$ enquanto que a produção ideal consistiu num valor médio de $250,48W$, sendo apenas $12,52\%$ da potência máxima que o aerogerador pode produzir, $2000W$.

Ao observar a potência perdida, relativamente à potência elétrica ideal (Figura H.2), justifica-se a aplicação de controlo como forma de tentar reduzir a mesma, uma vez que a produção real corresponde aproximadamente $5,9\%$ da potência ideal gerada.
Aplicado o controlador difuso com inferência de Mamdani, de forma a controlar a direção do aerogerador (Figura H.3), a produção estimada foi $14,28\%$ superior à produção real

Figura 5.26: Vento quase estacionário - Direção do vento no intervalo [100000;108000] amostras [Filtrado], Ta = 0,8 segundos.

Direção do vento estacionário (filtrado) Direção do vento real

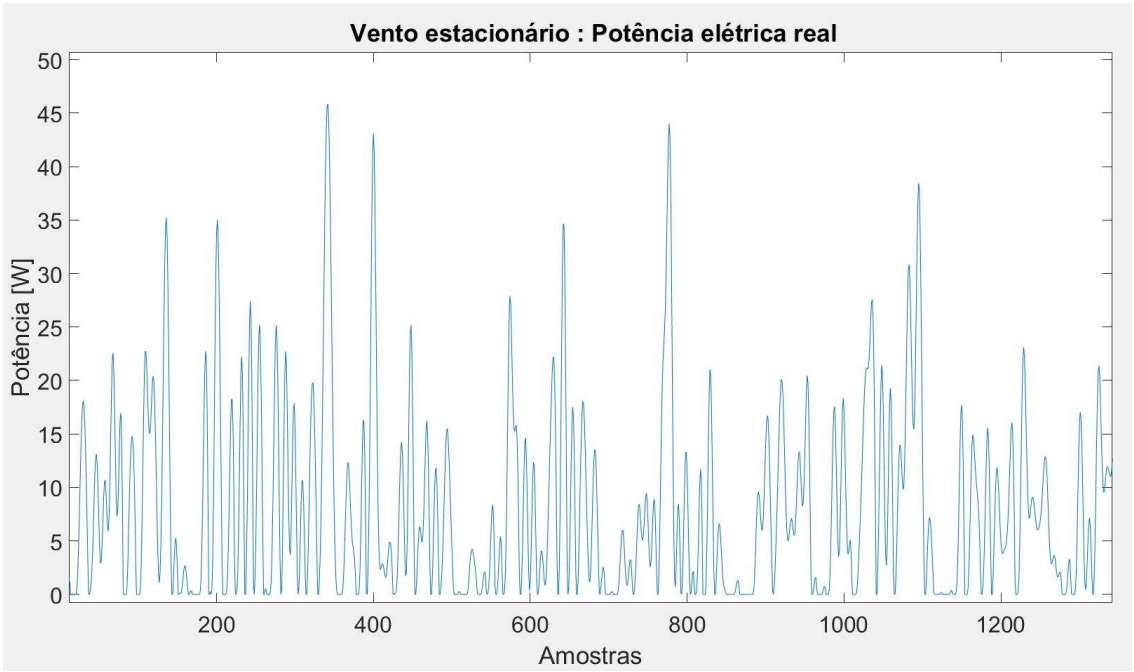


Figura 5.27: Vento quase estacionário - Potência real no intervalo [100000;108000] amostras, Ta = 0,8 segundos.

Potência real

(Figura 5.28), em média $16,90W$, representando 6,75% da produção ideal.
Tal como esperado, em condições em que não há grandes variações na velocidade do vento, a potência elétrica produzida com o controlador poderia ser superior à real, uma vez que existe um desalinhamento natural entre o aerogerador e a direção do vento, proporcionando uma perda constante de energia. Com a aplicação do controlador, este força o aerogerador a manter-se na direção do vento, contrariando as suas limitações físicas.

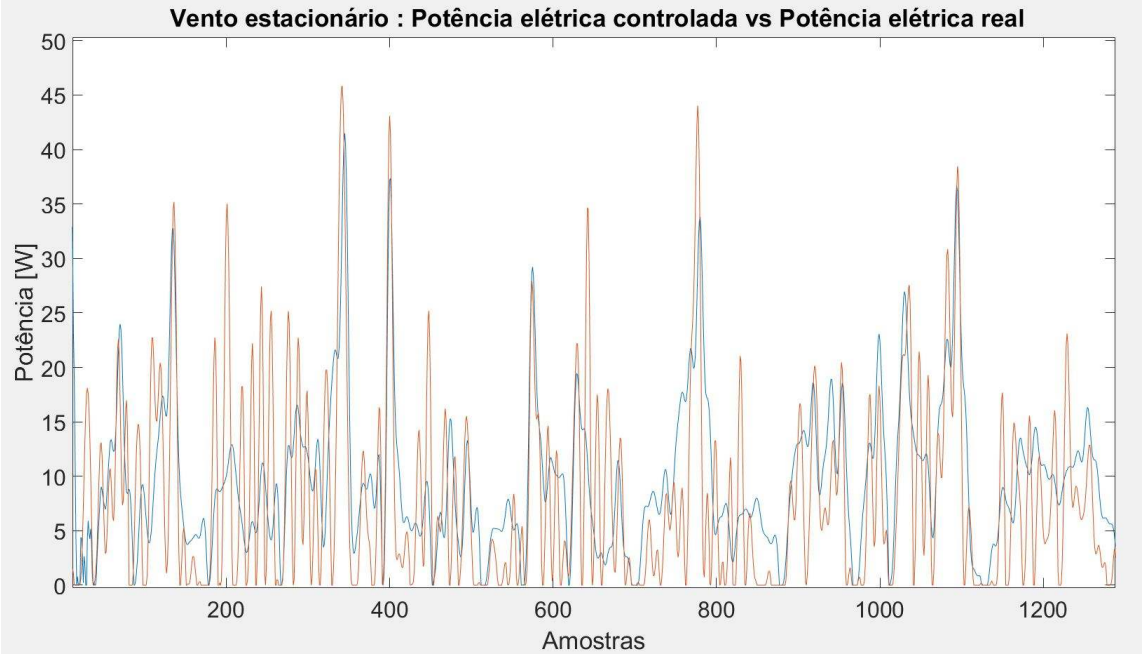


Figura 5.28: Vento quase estacionário - Potência produzida ao ser aplicado o controlador, Ta = 0,8 segundos.
Potência elétrica controlada Potência elétrica real

Os resultados desta experiência estão representados na Tabela 5.2, em que **Pc** é a percentagem da potência ideal produzida com a aplicação do controlador e **Pr** a percentagem da potência ideal produzida sem a aplicação do controlador.

Tabela 5.2: Resultados associados a experiência com o vento quase estacionário.

Modelos/Produção	**Potência gerada [W]**	**Pr[%]**	**Pc[%]**
Modelo Real	14	5,9	6,75
Modelo Ideal	250	-	-
Modelo Real com Controlador	16,90	-	-

5.5.2 Simulação - Rajadas de vento

Considerando a condição em que existiam rajadas de vento, foram escolhidas 8000 amostras no intervalo [32000;40000], Figura 5.29.

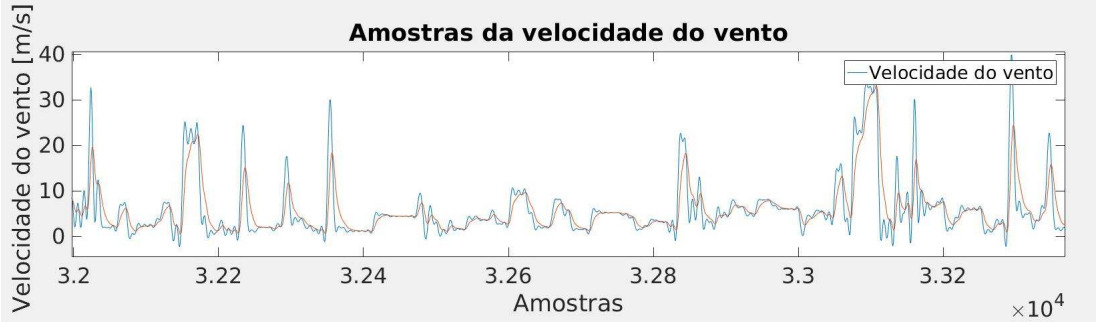


Figura 5.29: Rajadas de vento - Velocidade do vento no intervalo [32000;33200] amostras, Ta = 0,8 segundos.
Velocidade do vento. Velocidade do vento filtrado.

À semelhança da condição do vento quase estacionário, foi observada a produção real nesse intervalo de amostras, Figura 5.30, tendo sido posteriormente calculada a produção ideal, Figura H.4.

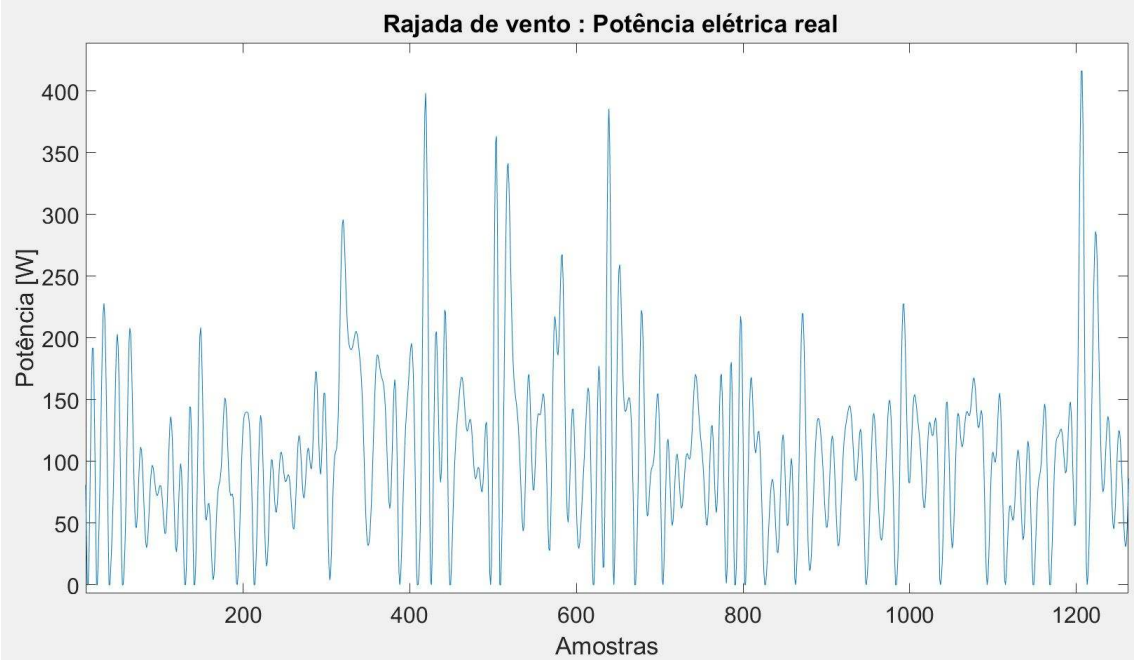


Figura 5.30: Rajadas de vento - Potência real produzida no intervalo [32000;33200] amostras, Ta = 0,8 segundos.
Potência elétrica real

A produção real foi em média $97,12W$, muito reduzida quando comparada à produção ideal, $371,71W$. Apesar de elevada, a produção ideal correspondia a 6,75% da potência máxima que pode ser produzida pelo aerogerador.

Considerando a potência perdida, Figura H.5, o mesmo controlador foi aplicado de forma a tentar reduzir as perdas, que neste caso consistia em aproximadamente 26,13% da potência gerada idealmente, $97,12W$. Após a aplicação do controlador,(Figura H.6), foi observado um aumento na produção estimada face à produção real de 4,42%, $101,42W$, Figura 5.31, representando 27,29% da produção ideal.

Um aumento na produção era esperado, uma vez que o vento afeta a produção com um fator de 3. Nesta condição em que o controlador mantinha o aerogerador mais próximo da direção do vento, este conseguia extrair mais potência do vento, gerando maior potência elétrica.

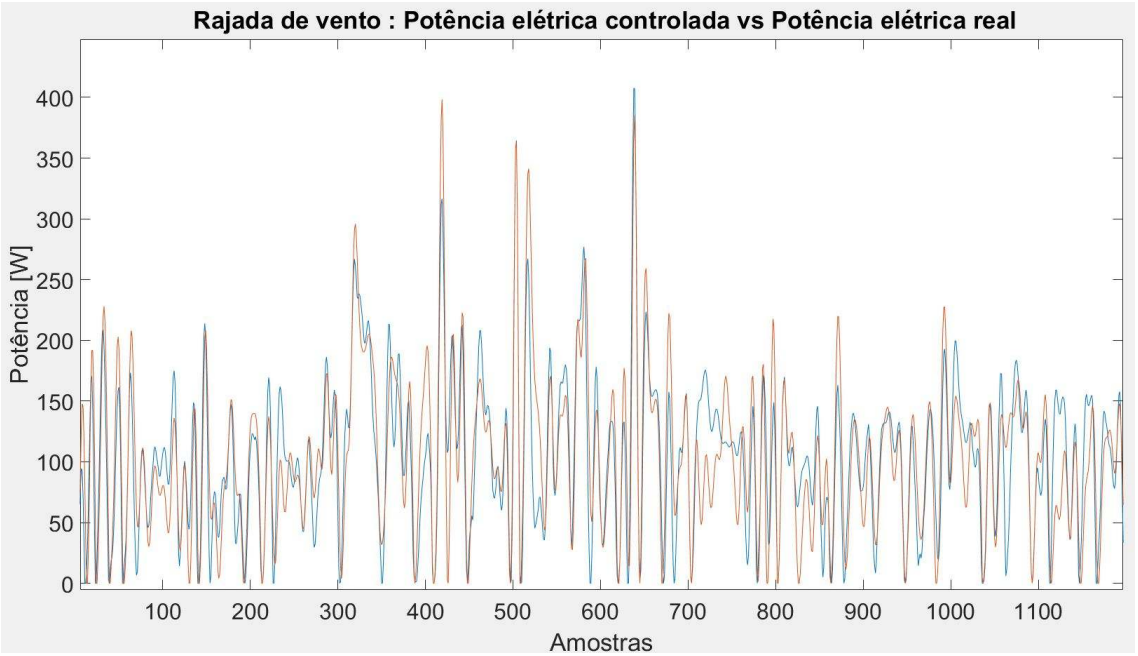


Figura 5.31: Rajadas de vento - Potência produzida ao ser aplicado o controlador, Ta = 0,8 segundos.

Potência elétrica controlada Potência elétrica real

Os resultados desta experiência estão resumidos na Tabela 5.3, em que **Pc** é a percentagem da potência ideal produzida com a aplicação do controlador e **Pr** a percentagem da potência ideal produzida sem a aplicação do controlador.

Tabela 5.3: Resultados associados a experiência com rajadas de vento.

Modelos/Produção	**Potência gerada [W]**	**Pr [%]**	**Pc [%]**
Modelo Real	97,12	26,13	27,29
Modelo Ideal	371,71	-	-
Modelo Real com Controlador	101,42	-	-

5.5.3 Simulação - Mudanças rápidas da direção do vento

Nesta simulação foram escolhidas as amostras no intervalo [1;8001], Figura 5.32, onde existiam variações da direção do vento de aproximadamente 190° em 4 amostras, aproximadamente 3,2 segundos. À semelhança das simulações anteriores, foi inicialmente observada a produção real para o intervalo discriminado (Figura5.33) e estimada a produção ideal, Figura H.7.

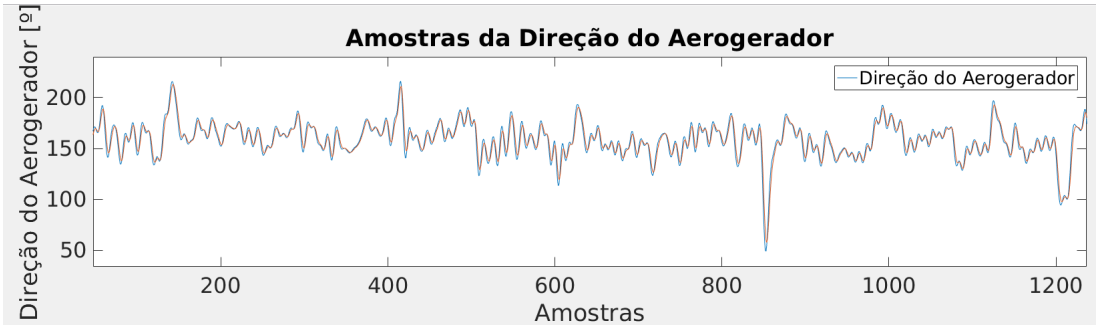


Figura 5.32: Mudanças rápidas da direção do vento - Direção do vento no intervalo de amostras [1;1200], Ta = 0,8 segundos .
Direção do vento. Direção do vento filtrado.

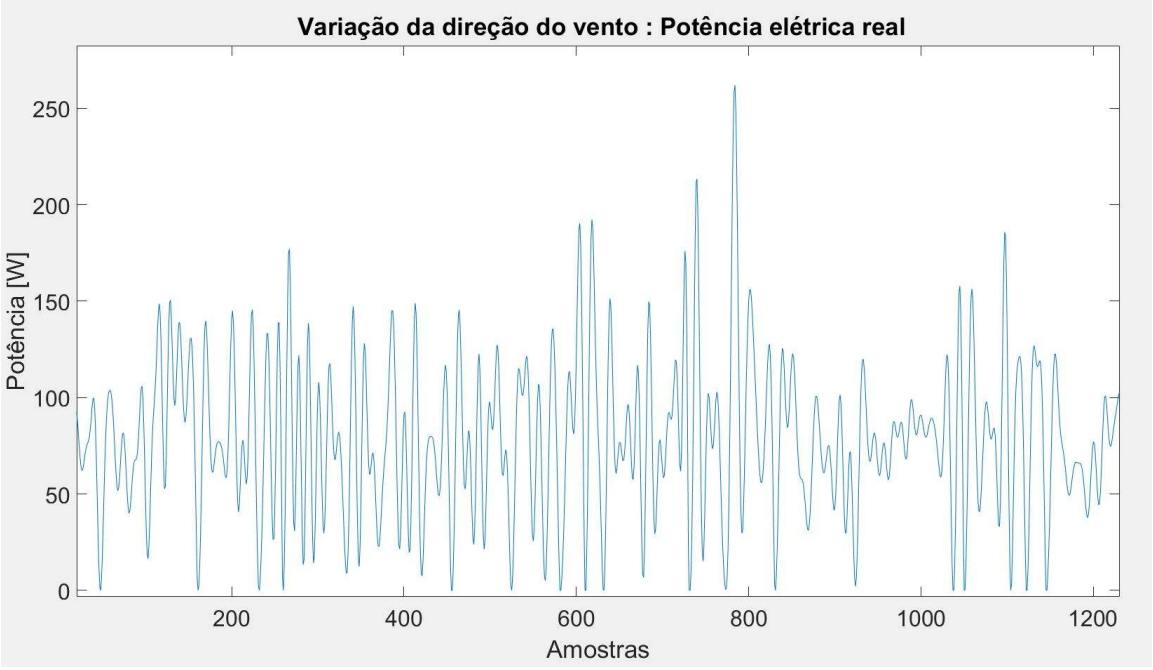


Figura 5.33: Mudanças rápidas da direção do vento - Potência real produzida no intervalo de amostras [1;1200], Ta = 0,8 segundos.
Potência elétrica real

Apesar da produção ideal ser elevada, quando comparada à produção real, consiste em apenas 6,75% da potência máxima que o aerogerador pode produzir. A produção real

no intervalo das 8000 amostras foi, em média, $83,61W$, enquanto que em condições ideais produziu $134,95W$.

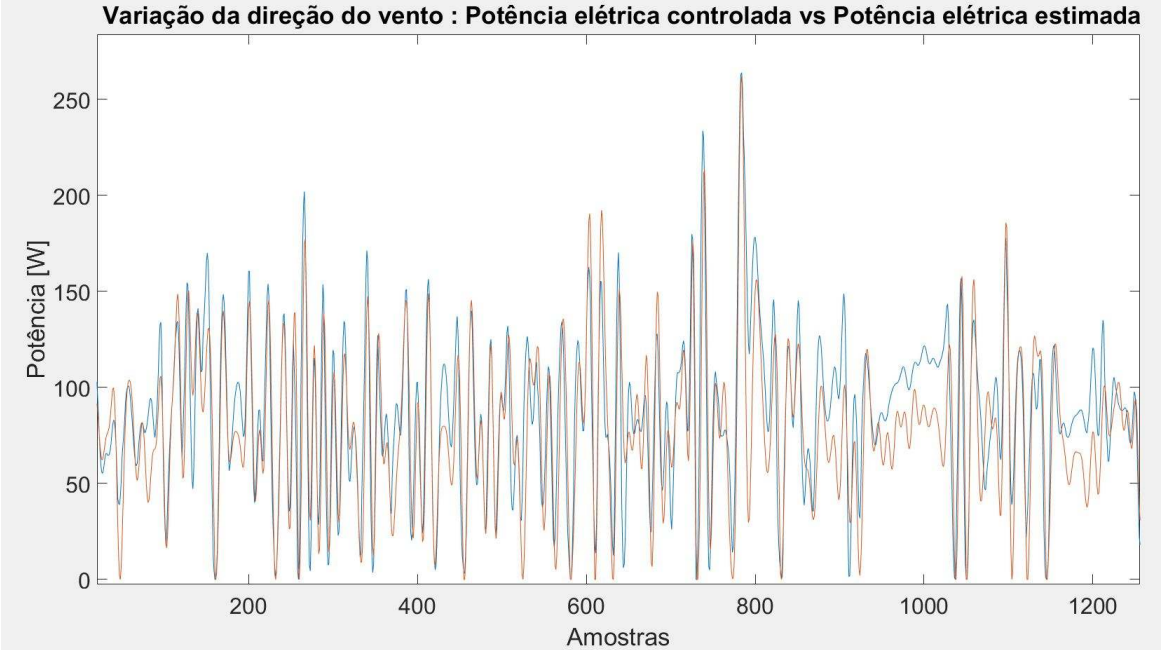


Figura 5.34: Mudanças rápidas da direção do vento - Potência produzida ao ser aplicado o controlador , Ta = 0,8 segundos.
Potencia elétrica controlada Potência elétrica real

A potência perdida pode ser observada na Figura H.8, sendo aproximadamente $61,89\%$ da potência ideal. Ao aplicar o controlador, (Figura H.9), foi observado um aumento na produção estimada em $5,11\%$ relativamente à produção real, Figura 5.34, $87,88W$ médios ($65,12\%$ da produção ideal), ao ser reduzido o erro de yaw .

Os resultados desta experiência estão representados na Tabela 5.4, em que **Pc** é a percentagem da potência ideal produzida com a aplicação do controlador e **Pr** a percentagem da potência ideal produzida sem a aplicação do controlador.

Tabela 5.4: Resultados associados a experiência com mudanças rápidas na direção vento.

Modelos/Produção	**Potência gerada [W]**	**Pr [%]**	**Pc [%]**
Modelo Real	83,61	61,89	65,12
Modelo Ideal	134,95	-	-
Modelo Real com Controlador	87,88	-	-

Através destas simulações, foi possível observar que a produção real tem sido limitada, uma vez que existem grandes diferenças entre esta e a produção ideal. Embora a produção ideal considere o C_p máximo alguma vez observado neste aerogerador e, o C_p varie ao longo do tempo com questões associadas com a aerodinâmica, a diferença na produção

deveria possuir uma relação superior à 60%. Um dos motivos para este acontecimento é o desalinhamento natural que existe neste aerogerador, face à direção do vento (segundo o ângulo de *Yaw*, Figura 5.12). Outro motivo pode ser observado na Figura 5.35, onde se verificam variações até aproximadamente 10° no ângulo de *Pitch*. Essas variações demonstram uma ligeira inclinação do aerogerador, contribuindo para o seu desalinhamento com a direção do vento. A conclusão retirada destes resultados é que, com o controlador, foi possível um aumento significativo da produção elétrica.

Com o controlo do ângulo de *Yaw* o aerogerador mantém-se aproximadamente na direção do vento, cumprindo o aumento da produção previsto com base na teoria. Este seguimento aproximado do vento proporcionado pelo controlador, considera a massa do aerogerador, com um valor de $60Kg$. Como tal, a ideia de que este deveria seguir totalmente a direção do vento não é de todo realista, uma vez que existem fatores como os atritos, área, massa e forças resistivas. Apesar da existência desses fatores, o erro de *Yaw* foi reduzido e, em condições de mudanças rápidas na direção do vento, que era a problemática desta dissertação, observou-se um aumento da produção de $5,11\%$. Considerada a corrente produzida pelo controlador de binário que atingia em condições extremas $2A$, esta é mínima quando comparada com os $5,8A$ máximos produzidos durante o período de aquisição de dados, o que torna a aplicação do controlador uma vantagem considerável, inclusive do ponto de vista energético.

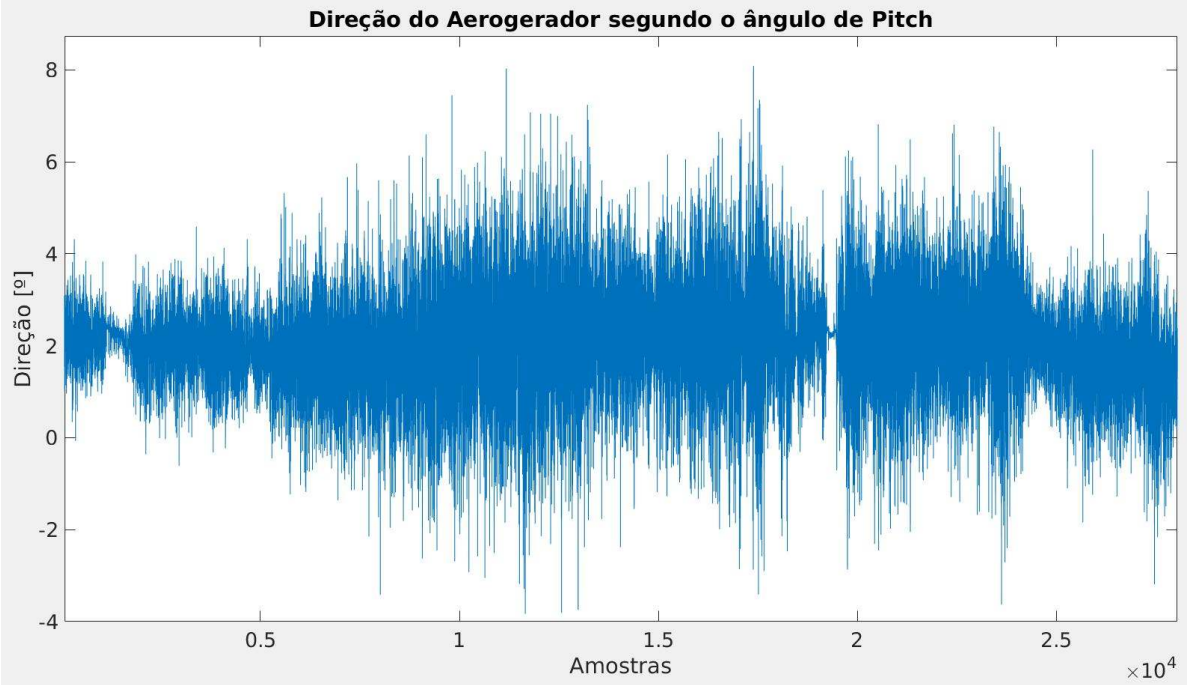


Figura 5.35: Variações no ângulo de *Pitch* do aerogerador, Ta = 0,8 segundos.
Direção do aerogerador (Pitch)

Os diagramas de blocos utilizados na simulação assim como todos os resultados e amostras utilizados, podem ser encontrados nos Anexos G e H consecutivamente.

Capítulo 6

Conclusões

Nesta secção são apresentadas as conclusões do trabalho realizado, Secção 6.1 e propostas para trabalhos futuros, Secção 6.2. São avaliadas as abordagens realizadas no processo de aquisição de dados, na modelação dos sistemas de produção elétrica e movimentação do aerogerador, no sistema de controlo aplicado e são discutidos os resultados obtidos.

6.1 Conclusões

Na realização desta dissertação não foram encontrados estudos energéticos e controlo aplicado aos aerogeradores de pequena potência, logo, os trabalhos relacionados serviram para reforçar a importância da estimação da produção e controlo nos sistemas de energia eólica, visto que o seu crescimento tem sido relevante.

Relativamente aos sistemas de aquisição de dados, estes cumpriram os seus propósitos. A linguagem de programação JAVA facilitou o processamento da receção e do envio de dados por rádio frequência, tornando-o dinâmico, na medida em que estes eram processados em paralelo. O armazenamento destes na base de dados também foi realizado com sucesso, contribuindo assim com uma redução do tempo de amostragem de aproximadamente 60 segundos, que existia previamente à implementação deste trabalho, para 4 segundos. Quanto à plataforma de monitorização, esta foi criada com sucesso, permitindo analisar as amostras obtidas e guardadas previamente na base de dados e, as organizar por data e hora. Esta organização por data e hora foi possível devida à implementação do sistema de sincronismo nos diversos métodos de aquisição de dados.

O objetivo principal desta dissertação era a criação de um modelo de potência elétrica real e de um modelo de potência elétrica ideal e, realização de uma análise comparativa

dos seus resultados, permitindo perceber a quantidade de energia perdida. Este objetivo foi cumprido e permitiu estimar a produção elétrica do aerogerador para qualquer valor de erro de *Yaw* e velocidade do vento. Com base nas potências elétricas estimadas foi possível observar que a potência real representava apenas 5,9% da potência ideal às baixas frequências de direção do vento, sendo que para as altas frequências de velocidade e direção do vento essas percentagens foram de 26,13% e 61,89% consecutivamente, existindo, portanto, uma perda significativa de energia relativamente a produção ideal.

Na situação das baixas frequências da direção do vento, este resultado pode ser explicado pelo facto do aerogerador possuir sempre um desalinhamento natural com a direção do vento, não aproveitando toda a potência do vento. Nestas mesmas condições, o modelo ideal considera um alinhamento com a direção vento e 40% da potência do vento (C_p), justificando a diferença entre os dois modelos de produção.

Quanto as altas frequências de velocidade e direção do vento o modelo real produz mais potência elétrica, uma vez que a intensidade da velocidade do vento é elevada, apesar do erro de Yaw não diferir muito da situação das baixas frequências. Nas mesmas condições, o modelo ideal atinge valores de produção de $2000W$, uma vez que considera mais potência do vento que o modelo real e um alinhamento otimo.

Outro objetivo desta dissertação era a criação de um controlador que permitisse reduzir o erro de Yaw e estimar a produção com este novo valor de erro, tendo sido também concretizado. Utilizando o esquema de controlo em cascata com os controladores de posição e de binário, o modelo de travão MR e o modelo de direção do aerogerador, foi possível verificar um aumento da produção elétrica de 14,28% nas baixas frequências da velocidade do vento e, nas altas frequências da velocidade e da direção do vento, verificou-se um aumento de 4,42% e 5,11% respetivamente.
Apesar dos resultados satisfatórios várias considerações foram feitas ao longo deste trabalho, incutindo assim incertezas em alguns resultados. Essas incertezas estão associadas a localização do aerogerador face aos sensores de medição das componentes do vento, vibrações e inclinações do aerogerador que influênciam os binários calculados, o coeficiente de potência e o rendimento elétrico que não são, na prática, constantes.

6.2 Proposta de trabalhos futuros

A realização desta dissertação poderá servir de mote ao estudo de vários aspetos, que poderão influenciar futuramente, tanto na produção como no tempo de vida do aerogerador em questão, produzir resultados interessantes quando considerados outros sistemas de produção eólica e até proporcionar novas implementações.

Algumas propostas para trabalhos futuros:

- Implementação do controlador ótimo preditivo de forma a melhorar o desempenho

do sistema de controlo;

- Investigação de forma a reduzir o tempo de amostragem na aquisição de dados e a criação de novos modelos;
- Realização de um estudo sobre os esforços mecânicas sofridos pelo aerogerador de pequena potência e, aplicação de controlo de forma a reduzi-los;
- Realização de um estudo comparativo da produção elétrica de uma VAWT face ao aerogerador modelado, considerando a localização no Departamento de Engenharia Eletrotécnica, assim como as condições observadas no presente trabalho.

Bibliografia

Afonso, M. A. (2010). "Sistema de Monitorização de Condições Meteorológicas e Correlação com Produção Renovável de Energia". Tese de mestrado. Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia. ISBN: 9241544902.

Almada Informa (2017). *Carta Hipsométrica do Concelho de Almada*. URL: http://www.m-almada.pt/portal/page/portal/AMBIENTE/AMB{_}NAT{_}BIO/?amb=0{\&}ambiente{_}ambiente{_}bio=12757660{\&}cboui=12757660.

Arduino (2016). *Arduino Uno*. URL: https://www.arduino.cc/en/Guide/Introduction.

Arduino (2017). *Arduino UNO and Genuino UNO*. URL: https://www.arduino.cc/en/main/arduinoBoardUno.

Ave, B. (2013). *MPU-6000 and MPU-6050 Product Specification*. Vol. 1. 408, pp. 1–17.

Buffard, B, L Chamerois e T. France (2014). "Energy Production Estimation of Faulty Wind Turbines". Em: (45), pp. 11–17.

Cactus.io (2016). *How to Hookup Davis Anemometer to Arduino*. URL: http://cactus.io/hookups/weather/anemometer/davis/hookup-arduino-to-davis-anemometer.

Castro, R. M. G. (2005). *Introdução à energia eólica*. Vol. 2005. 2.1. Universidade Técnica de Lisboa Instituto Superior Técnico DEEC / Secção de Energia Energias, p. 38. DOI: http://inerte.horabsurda.org/wp-content/uploads/Introducao_a_Energia_Fotovoltaica.pdf. URL: http://www.troquedeenergia.com/Produtos/LogosDocumentos/Introducao{_}a{_}Energia{_}Mini-Hidrica.pdf.

Chiodi, F. (2016). *Wind Turbine*. URL: http://www.123dapp.com/smb-123D{_}Design/Wind-Turbine/2150983.

CM-Moura (2016). *CM-Moura*. URL: http://www.cm-moura.pt/feirasdeamareleja/CentralFotovoltaica.html.

Costa, E. L. d., F. G. d. Lima e M. J. T. Ferreira A. A. d. A.Ferracini (2014). "Desenvolvimento de um controlador de tensã para um micro aerogerador para utilização em região urbana". Tese de doutoramento. Universidade Tecnológica Federal do Paraná.

Dariush Ghorbany (2011). "MR Damper hysteresis characterization for the semi-active suspension system". Tese de doutoramento. University of Agder, pp. 1–65.

Darus, R (2008). "Modeling and Control of Active Suspension for a Full Car". Tese de doutoramento. University Teknologi Malaysia.

Data, T. (2008). "MRF-140CG Magneto-Rheological Fluid". Em: *Lord product selector guide: lord magnetorheological fluids*. Vol. 74. LORD Corporation, pp. 5–6. DOI: ODDS7012(Rev.17/08). URL: www.lord.com.

Davis Instruments (2013). "Input / Output Connections". Em: vol. 2778, pp. 1–2.

Decagon Devices, I. (2016). *Davis Cup Anemometer*.

DFROBOT (2016). *APC220 Radio Data Module(SKU:TEL0005)*. URL: https://www.dfrobot.com/wiki/index.php/APC220{_}Radio{_}Data{_}Module(SKU:TEL0005){\#}Specification.

EDP (2016). *EDP Renováveis*. URL: http://www.edp.pt/pt/aedp/unidadesdenegocio/energiasrenovaveis/Pages/EnergiasRenovaveis.aspx.

ELECTROFUN (2016). *Módulo RF Transmissor e Receptor 433Mhz AM para Arduino e Raspberry PI*. URL: https://www.electrofun.pt/modulo-rf-transmissor-receptor.

Eltra (2014). "EL - ER 58 B / C / H / T Incremental Encoder". Em: *Eltra* 1, pp. 7–9.

Ferreira, A. A. d. A. (2011). "Sistema de produção de energia eólica". Tese de doutoramento. Faculdade de Engenharia da Universidade do Porto, pp. 14–16.

FILIPEFLOP (2016). *Comunicação Wireless com Módulo RF 433MHZ*. URL: http://blog.filipeflop.com/wireless/modulo-rf-transmissor-receptor-433mhz-arduino.html.

Florin, A. e P. Liliana (2013). "Pasive Suspension Modeling Using Matlab , Quarter Car Model , Input Signal Step Type". Em: *TEHNOMUS - New Technologies and Products in Machine Manufacturing Technologies*(1224-029x), pp. 258–263.

Gebraad, P. M. O., F. W. Teeuwisse, J. W. Van Wingerden, P. A. Fleming, S. D. Ruben, J. R. Marden e L. Y. Pao (2014). "A data-driven model for wind plant power optimization by yaw control". Em: *Proceedings of the American Control Conference*, pp. 3128–3134. ISSN: 07431619. DOI: 10.1109/ACC.2014.6859118.

Gil, P. (2013). *Controlo Inteligente*.

Github (2017). *minimu-9-ahrs-arduino*. URL: https://github.com/pololu/minimu-9-ahrs-arduino (acedido em 24/01/2017).

Jonkman, B. J. (2009). "TurbSim User ' s Guide : TurbSim User ' s Guide :" em: *Renewable Energy*(September).

Kaldellis, J. K. e D. Zafirakis (2011). "The wind energy (r)evolution: A short review of a long history". Em: *Renewable Energy* 36(7), pp. 1887–1901. ISSN: 09601481. DOI: 10.1016/j.renene.2011.01.002.

Karakoc, K. (2007). "Design of a magnetorheological brake system based on magnetic circuit optimization master of applied science". Tese de doutoramento. University of Victoria, pp. 17–103.

Kragh, K. A. e P. A. Fleming (2012). "Rotor speed dependent yaw control of wind turbines based on empirical data". Em: *Proceedings of 50th AIAA Aerospace Sciences {...}*(January), pp. 1–3. DOI: 10.2514/6.2012-1018. URL: http://arc.aiaa.org/doi/pdf/10.2514/6.2012-1018.

Laboratory, A. S. (2016). *Active Structures Laboratory*. URL: http://scmero.ulb.ac.be/project.php?id=9{\&}page=MR{_}rehabilitation{_}design.html.

Lajqi, S. e S. Pehan (2012). "Designs and optimizations of active and semi-active nonlinear suspension systems for a terrain vehicle". Em: *Strojniski Vestnik/Journal of Mechanical Engineering* 58(12), pp. 732–743. ISSN: 00392480. DOI: 10.5545/sv-jme.2012.776.

Leitão, N. (2012). "Medição Experimental do Coeficiente de Resistência ao Rolamento e do Coeficiente de Arrasto Aerodinâmico de um Veículo Automóvel de Elevada Eficiência Energética". Em:

LORD Corporation. *LORD Corporation - MRF*. URL: http://www.lordmrstore.com/lord-mr-products/mrf-140cg-magneto-rheological-fluid.

MathWorks (2016). *Sugeno-Type Fuzzy Inference*. URL: http://www.mathworks.com/help/fuzzy/what-is-sugeno-type-fuzzy-inference.html.

MathWorks (2017a). *Fluke Instruments and MATLAB*. URL: https://www.mathworks.com/products/instrument/supported/fluke.html.

MathWorks (2017b). *Improve Neural Network Generalization and Avoid Overfitting*. URL: https://www.mathworks.com/help/nnet/ug/improve-neural-network-

generalization - and - avoid - overfitting . html ? requestedDomain = www . mathworks.com.

Melorose, J, R Perroy e S Careas (2015). "Arduino UNO". Em: *Statew. Agric. L. Use Baseline 2015* 1. ISSN: 1098-6596. DOI: 10 . 1017 / CBO9781107415324 . 004. arXiv: arXiv : 1011 . 1669v3.

Moura, E. D. A. (2003). "Estudo de Suspensões Passiva, Semi-Ativa MR e Ativa". Tese de doutoramento. Universidade Federal de Itajubá, p. 170.

National Instruments (2016). *National Instruments*. URL: http : / / www . ni . com / tutorial / 7109 / pt /].

NetBeans (2016). *Connecting to a MySQL Database*. URL: https : / / netbeans . org / kb / docs / ide / mysql . html.

Network, S. e S. Republic (2000). *Incremental Encoders*. Eltra.

NXP Semiconductors (2014). "UM10204 I²C-bus specification and user manual". Em: (April), p. 64. arXiv: UM10204.

Oliveira, K. F. (2015). "Controlo semi-ativo da suspensão de um veículo automóvel". Tese de doutoramento. Instituto Polit{é}cnico de Bragan{ç}a, pp. 9–12.

Paré, C. A., D. J. Nelson, W. R. Saunders e K. Semiactive (1998). "Experimental Evaluation of Semiactive Magneto- Rheological Suspensions for Passenger Vehicles". Tese de doutoramento. Faculty of the Virginia Polytechnic Institute e State University.

Park, E. J., D. Stoikov, L. Falcao da Luz e A. Suleman (2006). "A performance evaluation of an automotive magnetorheological brake design with a sliding mode controller". Em: *Mechatronics* 16(7), pp. 405–416. ISSN: 09574158. DOI: 10 . 1016 / j . mechatronics . 2006 . 03 . 004.

Paschoal, E. F. (2011). "Controle Semi-ativo de Vibrações Usando Lógica Nebulosa e Fluido Magnetoreológico". Tese de doutoramento. Universidade Estadual Paulista.

Paulo Gil (2015). *Sistemas de Controlo*. URL: http : / / lightenjin . pt / lightenjin - gesluce - dali / {\$}{\%}5C{\$}n.

PJRC (2016). *VirtualWire Library*. URL: https : / / www . pjrc . com / teensy / td{_}libs{_ }VirtualWire . html.

Pololu (2017). *AltIMU-10 v4*. URL: https : / / www . pololu . com / product / 2470 / resources (acedido em 24/01/2017).

Poznić, A, A Zelić e L Szabó (2012). "Magnetorheological Fluid Brake–Basic Performances Testing with Magnetic Field Efficiency Improvement Proposal". Em: *Hungarian Journal of Industry and Chemistry* 40(2), pp. 113–119. URL: http://mkweb.uni-pannon.hu/hjic/index.php/hjic/article/view/351.

PTROBOTICS (2016). *APC220 Radio Communication Module*. URL: http://www.ptrobotics.com/wireless/2570-apc220-radio-communication-module.html.

Reddy, P. R. K., K. M. Rao e P. B. Kishore (2015). "Wind Turbine Pitch and Yaw Control". Em: *International Journal of Science, Technology & Management*(04), pp. 618–623.

REDSTONE (2008). *Black-box vs White-box Testing: Choosing the Right Approach to Deliver Quality Applications*. URL: http://www.cs.unh.edu/{~}it666/reading{_}list/Defense/blackbox{_}vs{_}whitebox{_}testing.pdf.

Roballo, S. T., G. Fisch, R. Da, M. Girardi, I. Cptec e S. J. Campos-sp (2009). "Escoamento Atmosférico No Centro De Lançamento De Alcântara (Cla): Parte Ii - Ensaios No Túnel De Vento". Em: *Rev. Bras. Meteorol.* 24(1), pp. 87–99.

Ródenas, L. (2016). *Arduino sketch that returns calibration offsets for MPU6050*.

Rossa, C., A. Jaegy, J. Lozada e A. Micaelli (2014). "Design considerations for magnetorheological brakes". Em: *IEEE/ASME Transactions on Mechatronics* 19(5), pp. 1669–1680. ISSN: 10834435. DOI: 10.1109/TMECH.2013.2291966.

Rowell, D. (2002). "State-space representation of lti systems". Em: *URL: http://web. mit. edu/2.14/www/Handouts/StateSpace. pdf*(October), pp. 1–6. URL: http://www.web.mit.edu/2.14/www/Handouts/StateSpace.pdfhttp://files/1757/StateSpace.pdf.

Rüncos, F., R. Carlson, P. Kuo-Peng, H. Voltolini e N. J. Batistela (2005). "Geração de energia eólica – tecnologias atuais e futuras". Em: *Rev. Eletr. Mod.*(figura 1), pp. 1–15. URL: http://ecatalog.weg.net/files/wegnet/WEG-geracao-de-energia-eolica-tecnologias-atuais-e-futuras-artigo-tecnico-portugues-br.pdf.

RXTX (2016). *Two way communcation with the serial port*. URL: http://rxtx.qbang.org/wiki/index.php/Two{_}way{_}communcation{_}with{_}the{_}serial{_}port.

Scholbrock, A., P. Fleming, A. Wright, Chris Slinger, J. Medley e M. Harris (2015). "Field test results from lidar measured yaw control for improved yaw alignment with the NREL Controls Advanced Research Turbine". Em: *AIAA SciTech*(January).

Shuqin, L. (2011). "Magnetic Suspension and Self-pitch for Vertical-axis Wind Turbines". Em: *Fundamental and Advanced Topics in Wind Power*. DOI: 16250. URL: http://www.

intechopen.com/books/export/citation/EndNote/fundamental-and-advanced-topics-in-wind-power/magnetic-suspension-and-self-pitch-for-vertical-axis-wind-turbines.

Standard, V.-c. (2004). *User 's Manual*. July. Yangzhou Shenzhou Wind-driven Generator Co.,Ltd, p. 3. ISBN: 9781119978152. DOI: 1558407.

STMicroeletronics (2012). "Ultra compact high performance e-Compass 3D accelerometer and 3D magnetometer module". Em: (June), p. 54. URL: https://www.pololu.com/file/download/LSM303D.pdf?file{_}id=0J703.

STMicroeletronics (2013). "Datasheet L3GD20H MEMS motion sensor: three-axis digital output gyroscope". Em: *STMicroeletronics*(March), pp. 1–52.

Stol, K (2002). "Dynamics modeling and periodic control of horizontal-axis wind turbines". Em: *PhD Thesis*,(JULY 2001).

Takagi, T. e M. Sugeno (1985). "Fuzzy Identification of Systems and Its Applications to Modeling and Control". Em: *IEEE Trans. on Systems, Man, and Cybernetics* 15(1), pp. 116–132. ISSN: 0018-9472.

Tempo (2017). *Histórico do tempo para Almada*. URL: https://www.tempo.pt/almada-sactual.htm.

Vladislavleva, E., T. Friedrich, F. Neumann e M. Wagner (2013). "Predicting the energy output of wind farms based on weather data: Important variables and their correlation". Em: *Renewable Energy* 50, pp. 236–243. ISSN: 09601481. DOI: 10.1016/j.renene.2012.06.036. arXiv: 1109.1922.

Welch, G. e G. Bishop (2006). "An Introduction to the Kalman Filter". Em: *In Practice* 7(1), pp. 1–16. ISSN: 10069313. DOI: 10.1.1.117.6808. URL: http://old.shahed.ac.ir/references/kalman{\%}7B{_}{\%}7Dfilter{\%}7B{_}{\%}7Dnotes.pdfhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.6578{\%}7B{\&}{\%}7Drep=rep1{\%}7B{\&}{\%}7Dtype=pdf.

Zhao, W. e K. Stol (2007). "Individual Blade Pitch for Active Yaw Control of a Horizontal-Axis Wind Turbine". Em: *45th AIAA Aerospace Sciences Meeting and Exhibit*(January), pp. 8–11. DOI: 10.2514/6.2007-1022. URL: http://pdf.aiaa.org/preview/CDReadyMASM07{_}1064/PV2007{_}1022.pdf.

Apêndice A

Código do PC Central

```
1   // Thread responsável pela receção de dados
2  public static class SerialReader implements Runnable {
3
4      InputStream in;
5      boolean getIDM = true;
6      boolean getIDA = true;
7      boolean getIDP = true;
8
9      static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
10     static final String DB_URL = "jdbc:mysql://localhost:3306/mysql";
11     static final String USER = "root";
12     static final String PASS = "thesis123";
13     Connection conn = null;
14     Statement stmt = null;
15     String sql;
16     int index, index1, index2, index3, index4, index5, index6, index7, index8;
17     int sec = 0;
18     byte[] Buffer = new byte[256];
19     byte[] resetBuffer = new byte[256];
20     int indexChar = 0;
21     int ID = 1;
22     int sample = 0;
23     int IDm = 1;
24     int IDp = 1;
25     java.util.Date myDate;
26     Calendar calendar = Calendar.getInstance();
27
28     public SerialReader(InputStream in) {
29         this.in = in;
30     }
31 // Obtém o último ID de cada uma das tabelas da base de dados.
```

```
32
33      void getID(String IDx, String Table) throws ClassNotFoundException,
34       SQLException {
35
36          Class.forName("com.mysql.jdbc.Driver");
37          conn = DriverManager.getConnection(DB_URL, USER, PASS);
38          stmt = conn.createStatement();
39
40          sql = "SELECT " + IDx + " FROM " + Table;
41          PreparedStatement ps = conn.prepareStatement(sql);
42          ResultSet rsx = ps.executeQuery(sql);
43
44          if (IDx.equalsIgnoreCase("IDm")) {
45              while (rsx.next()) {
46                  IDm = rsx.getInt(IDx) + 1;
47              }
48          } else if (IDx.equalsIgnoreCase("ID")) {
49              while (rsx.next()) {
50                  ID = rsx.getInt(IDx) + 1;
51              }
52
53          } else if (IDx.equalsIgnoreCase("IDp")) {
54              while (rsx.next()) {
55                  IDp = rsx.getInt(IDx) + 1;
56              }
57
58          }
59
60          stmt.close();
61          conn.close();
62
63      }
64
65  @Override
66  public void run() {
67      byte[] buffer = new byte[256];
68      int numBytes = -1;
69      int indexTotal = 0;
70
71      try {
72          String data = "";
73          String[] dataArray = new String[30];
74
75          String dataString = "";
76          while (((numBytes = this.in.read(buffer)) > -1)) {
77
78              for (int i = 0; i < numBytes; i++) {
79
80                  if (indexTotal < 255 && i < 255) {
81
```

```
82                      Buffer[indexTotal] = buffer[i];
83  // Contém os dados lidos da porta
84
85                      if (Buffer[indexTotal] == '\n') {
86  // Começa a ler e dividir a String quando há um \n
87
88                          dataString = new String(Buffer);
89
90                          try {
91
92                              double pitch = 0, yaw = 0, velocityM = 0,
93                              positionM = 0, Power = 0, Voltage = 0,
94                               Current = 0;
95
96                              dataArray = dataString.split("\\s+");
97  // Divide a String em palavras separadas por espaço/tab/etc..
98                              if (dataArray[0].equals("S")) {
99   // Caso o receba dados do sensor acoplado ao aerogerador,
100  //valida a data e insere na tabela "Aerodata"
101                                 if (Integer.parseInt(dataArray[5]) != sec) {
102                                     sec = Integer.parseInt(dataArray[5]);
103                                     if (getIDA) {
104                                         getID("ID", "mysql.aerodata");
105                                         getIDA = false;
106                                     }
107                                     pitch = Double.parseDouble(dataArray[1]);
108
109                                     yaw = Double.parseDouble(dataArray[2]);
110                                     if (yaw != 0) {
111                                         yaw = 360 - yaw;
112                                     }
113
114                                     DateFormat formatter = new
115                                     SimpleDateFormat("dd/MM/yyyy␣HH:mm:ss");
116
117                                     String DateX = dataArray[6] + "/" +
118                                     dataArray[7] + "/" + dataArray[8] +
119                                     "␣" + dataArray[3] + ":" +
120                                      dataArray[4] + ":" + dataArray[5];
121
122                                     if (dateValidation(dataArray[6] + "-" +
123                                      dataArray[7] +"-" + dataArray[8])) {
124
125                                         myDate = formatter.parse(DateX);
126                                         java.sql.Date sqlDate = new
127                                         java.sql.Date(myDate.getTime());
128
129                                         java.sql.Time sqlTime = new
130                                         java.sql.Time(myDate.getTime());
131
```

```
132                                     sql = "INSERT␣INTO␣mysql.aerodata(ID," +
133                                     "Pitch," + "Yaw)" + "␣VALUE(" +
134                                     ID + "," + pitch + "," +
135                                     yaw + ")";
136
137                                     System.out.println("S␣" + pitch +
138                                     "␣" + yaw + "␣" +sqlDate.toString()
139                                     + "␣" + sqlTime.toString());
140
141                                     Class.forName("com.mysql.jdbc.Driver");
142
143                                     conn = DriverManager.
144                                     getConnection(DB_URL, USER, PASS);
145
146                                     stmt = conn.createStatement();
147                                     int rs = stmt.executeUpdate(sql);
148
149                                     PreparedStatement ps ;
150                                     ps=conn.prepareStatement("" +
151                                     "␣UPDATE␣mysql.aerodata␣SET␣" +
152                                     "␣Data=␣?,␣Hora=␣?␣␣WHERE␣ID=␣?");
153
154                                     ps.setDate(1, sqlDate);
155                                     ps.setTime(2, sqlTime);
156                                     ps.setInt(3, ID);
157                                     ps.executeUpdate();
158                                     stmt.close();
159                                     conn.close();
160                                     ID = ID + 1;
161                                 }
162                                 indexTotal = -1;
163                             }
164                         }
165                         if (dataArray[0].equals("M")) {
166 // Caso o receba dados da estação meteorológica, valida a data e
167 //insere na tabela "meteodata"
168                             if (getIDM) {
169                                 getID("IDm", "mysql.meteodata");
170                                 getIDM = false;
171
172                             }
173                             velocityM = (Double.parseDouble(
174                             dataArray[1]));
175
176                             if (velocityM > 32.7) {
177                                 velocityM = 32.7;
178                             }
179                             positionM = Double.
180                             parseDouble(dataArray[2]);
181
```

```
182                             DateFormat formatter = new
183                             SimpleDateFormat("dd/MM/yyyy␣HH:mm:ss");
184
185                             String DateX = dataArray[6] + "/" +
186                              dataArray[7] + "/" +dataArray[8] +
187                              "␣" + dataArray[3] + ":" +
188                               dataArray[4] + ":" + dataArray[5];
189
190                             if (dateValidation(dataArray[6] +
191                             "-" + dataArray[7] + "-" +
192                              dataArray[8])) {
193
194
195                                 myDate = formatter.parse(DateX);
196
197                                 java.sql.Date sqlDate = new
198                                 java.sql.Date(myDate.getTime());
199
200                                 java.sql.Time sqlTime = new
201                                 java.sql.Time(myDate.getTime());
202
203                                 sql = "INSERT␣INTO␣mysql." +
204                                 "meteodata(IDm,VelocidadeM," +
205                                  "PosicaoM)"+ "␣VALUE(" + IDm +
206                                   "," + velocityM + ","+
207                                   positionM + ")";
208
209                                 System.out.println("M␣" +
210                                 velocityM + "␣" + positionM + "␣" +
211
212                                  sqlDate.toString() + "␣" +
213                                  sqlTime.toString());
214
215
216                                 Class.forName("com.mysql.jdbc.Driver");
217                                 conn = DriverManager.getConnection
218                                 (DB_URL, USER, PASS);
219                                 stmt = conn.createStatement();
220
221                                 int rs = stmt.executeUpdate(sql);
222
223                                 PreparedStatement ps;
224                                 ps = conn.prepareStatement("UPDATE␣" +
225                                 "␣mysql.meteodata␣SET␣DataM=␣?␣,␣"+
226                                 "␣HoraM=␣?␣␣WHERE␣IDm=␣?");
227
228                                 ps.setDate(1, sqlDate);
229                                 ps.setTime(2, sqlTime);
230                                 ps.setInt(3, IDm);
231                                 ps.executeUpdate();
```

```
232                                     stmt.close();
233                                     conn.close();
234                                     IDm = IDm + 1;
235                                 }
236                                 indexTotal = -1;
237                             }
238                             if (dataArray[0].equals("P")) {
239 // Caso o receba dados da estação meteorológica, valida a data
240 // e insere na tabela "powerdata"
241
242                                 if (getIDP) {
243                                     getID("IDp", "mysql.powerdata");
244                                     getIDP = false;
245
246                                 }
247
248                                 DateFormat formatter;
249                                 formatter = new SimpleDateFormat("␣"+
250                                  "dd/MM/yyyy␣HH:mm:ss");
251
252                                 String DateX = dataArray[2] + "/" +
253                                 dataArray[3] + "/" + dataArray[4] +
254                                  "␣" + dataArray[5] + ":" + dataArray[6] +
255                                  ":" + dataArray[7];
256
257                                 Power = Math.abs(Double.
258                                 parseDouble(dataArray[1]));
259
260                                 if (dateValidation(dataArray[2] + "-" +
261                                  dataArray[3] + "-" + dataArray[4])) {
262
263                                     myDate = formatter.parse(DateX);
264                                     java.sql.Date sqlDate = new java.sql.
265                                     Date(myDate.getTime());
266
267                                     java.sql.Time sqlTime = new java.sql.
268                                     Time(myDate.getTime());
269
270                                     sql = "INSERT␣INTO␣mysql.powerdata␣" +
271                                     "(IDp,Potencia)" + "␣VALUE(" +
272                                     IDp + "," + Power + ")";
273
274                                     System.out.println("P␣" + Power +
275                                      "␣" + sqlDate.toString() + "␣" +
276                                      sqlTime.toString());
277
278                                     Class.forName("com.mysql.jdbc.Driver");
279                                     conn = DriverManager.
280                                     getConnection(DB_URL, USER, PASS);
281                                     stmt = conn.createStatement();
```

```
282                                     int rs = stmt.executeUpdate(sql);
283                                     PreparedStatement ps;
284                                     ps = conn.prepareStatement("UPDATE " +
285                                     " mysql.powerdata SET DataP= ? , "+
286                                     " HoraP= ?  WHERE IDp= ?");
287
288                                     ps.setDate(1, sqlDate);
289                                     ps.setTime(2, sqlTime);
290                                     ps.setInt(3, IDp);
291                                     ps.executeUpdate();
292                                     stmt.close();
293                                     conn.close();
294                                     IDp = IDp + 1;
295                                 }
296                                 indexTotal = -1;
297                             }
298                         } catch (SQLException e) {
299                         } catch (NumberFormatException e) {
300                             System.out.println("Not a number");
301                         } catch (ClassNotFoundException ex) {
302                             Logger.getLogger(CentralComm.class.
303                             getName()).log(Level.SEVERE, null, ex);
304                         } catch (ParseException ex) {
305                             Logger.getLogger(CentralComm.class.
306                             getName()).log(Level.SEVERE, null, ex);
307                         }
308                         indexTotal = -1;
309                     }
310                     indexTotal = indexTotal + 1;
311                 } else {
312                     indexTotal = 0;
313                 }
314             }
315         }
316
317     } catch (IOException e) {
318         e.printStackTrace();
319     }
320 }
321 // Calcula o fator de potência.
322
323 double powerFactor(int sample, double[] vectorVoltage,
324 double[] vectorCurrent) {
325
326     double instantPower = 0, sum_instantPower = 0;
327     double realPower = 0, sum_squared_Voltage = 0;
328     double sum_squared_Current = 0, squared_Current;
329     double root_mean_square_voltage, mean_square_voltage;
330     for (int n = 0; n < sample; n++) {
331
```

```
332         instantPower = vectorVoltage[n] * vectorCurrent[n];
333         sum_instantPower += instantPower;
334     }
335 
336     realPower = sum_instantPower / sample;
337     double squared_Voltage = 0;
338     for (int n = 0; n < sample; n++) {
339         squared_Voltage = vectorVoltage[n] * vectorVoltage[n];
340 
341         sum_squared_Voltage += squared_Voltage;
342     }
343 
344     mean_square_voltage = sum_squared_Voltage / sample;
345     root_mean_square_voltage = Math.sqrt(mean_square_voltage);
346 
347     for (int n = 0; n < sample; n++) {
348 
349         squared_Current = vectorCurrent[n] * vectorCurrent[n];
350         sum_squared_Current += squared_Current;
351     }
352     double mean_square_current = sum_squared_Current / sample;
353     double root_mean_square_current = Math.sqrt(mean_square_current);
354     double apparentPower =root_mean_square_voltage*root_mean_square_current;
355     double power_factor = realPower / apparentPower;
356     return power_factor;
357 }
358 
359         // Valida a data: "true " se esta é valida, "false" se não é válida.
360 
361 boolean dateValidation(String date) {
362 //stackoverflow.com/questions/226910/how-to-sanity-check-a-date-in-java ,
363 
364     try {
365         DateFormat df = new SimpleDateFormat(DATE_FORMAT);
366         df.setLenient(false);
367         df.parse(date);
368         return true;
369     } catch (ParseException e) {
370         return false;
371     }
372 }
373 }
374 
375 // Thread responsável pela transmissão de dados de sincronismo
376 public static class SerialWriter implements Runnable {
377 
378     OutputStream out;
379     long dFinal;
380     boolean sync = true;
381     Calendar calendar = Calendar.getInstance();
```

```
382     long dInitial = calendar.getTimeInMillis();
383 
384     public SerialWriter(OutputStream out) {
385         this.out = out;
386     }
387 
388     public void run() {
389         try {
390 
391             int c = 0;
392             long total = 0;
393             String messageString = "";
394 
395             while (sync == true) {
396  //Obtém os dados de Data e Hora do sistema e de
397  //60 em 60 segundos envia em broadcast
398                 calendar = Calendar.getInstance();
399                 dFinal = calendar.getTimeInMillis();
400                 total = dFinal - dInitial;
401 
402                 if (total > 60000) {
403                     TimeUnit.MILLISECONDS.sleep(500);
404 
405                     messageString = "␣D␣" +
406                     (calendar.getTimeInMillis() / 1000)
407                     + "␣";
408 
409                     System.out.println(messageString);
410                     this.out.write(messageString.
411                     getBytes());
412 
413                     dInitial = calendar.getTimeInMillis();
414                     TimeUnit.MILLISECONDS.sleep(500);
415                 }
416             }
417         } catch (IOException e) {
418             e.printStackTrace();
419         } catch (InterruptedException ex) {
420             Logger.getLogger(CentralComm.class.getName()).
421             log(Level.SEVERE, null, ex);
422         }
423     }
424 }
```

APÊNDICE B

Código da Plataforma de Monitorização

```
1
2  String tables = " and mysql.aerodata.Hora = "+
3   "mysql.meteodata.HoraM GROUP BY mysql.aerodata.`Hora`";
4
5     static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
6     static final String DB_URL = "jdbc:mysql://localhost:3306/mysql";
7     static final String USER = "root";
8     static final String PASS = "thesis123";
9     Connection conn = null;
10    Statement stmt = null;
11    String sql;
12    ResultSet rs;
13    XYSeries GoalsA = new XYSeries("Velocidade do Aerogerador[m/s]");
14    XYSeries GoalsA2 = new XYSeries("Posicao  do Aerogerador[Rad]");
15    XYSeries GoalsM = new XYSeries("Velocidade do Vento[m/s]");
16    XYSeries GoalsM2 = new XYSeries("Posicao do Vento[Rad]");
17
18    XYSeriesCollection xyDataset = new XYSeriesCollection();
19    JFreeChart chart = null;
20    JFreeChart chart2 = null;
21
22    Calendar cCal = null;
23
24    public Interface() {
25
26        CreateChart();
27        initComponents();
28    }
29
30     void CreateChart() {
31
```

```
32          // Goals.add(0, 0);
33          chart = ChartFactory.createXYLineChart(
34                  "Dados␣do␣Vento", "Minuto", "Velocidade/Direcao␣do␣Vento",
35                  xyDataset, PlotOrientation.VERTICAL, true, true, false);
36
37          ChartPanel cp = new ChartPanel(chart) {
38
39              public Dimension getPreferredSize() {
40                  return new Dimension(605, 508);
41              }
42          }
43
44          add(cp);
45          chart.getXYPlot().setBackgroundPaint(Color.BLACK);
46          setDefaultCloseOperation(EXIT_ON_CLOSE);
47          pack();
48      }
49  //Limpa os TextFields relativos a data.
50
51      void limparData() {
52          jTextField3.setText("");
53          jTextField4.setText("");
54          jTextField5.setText("");
55          d = false;
56          data = "";
57      }
58
59  //Limpa o TextField relativo a hora.
60      void limparHora() {
61          h = false;
62          jTextField6.setText("");
63          hora = "";
64      }
65  //Seleciona da Base de dados e adiciona ao gráfico os dados de Velocidade e
66  //Direcao do vento e do aerogerador que tenham a data e hora em comum.
67
68      void MixedH(int op, ResultSet rsx, java.sql.Time Time, double YA,
69       double YA2, double YM, double YM2, DateFormat formatter)
70        throws SQLException, ParseException {
71
72          while (rsx.next()) {
73              if (rsx.getDate("Data") == null || rsx.getDate("DataM") == null) {
74                  String result = rsx.getDouble("Velocidade") + "\t\t" +
75                  rsx.getDouble("Posicao")
76
77                  + "\t\t" + rsx.getDouble("VelocidadeM") + "\t\t" +
78                  rsx.getDouble("PosicaoM") + "\t\t␣-␣\t\t␣-␣\n";
79
80                  textArea1.append(result);
81
```

```
82              } else {
83                  YM = rsx.getDouble("VelocidadeM");
84                  YM2 = rsx.getDouble("PosicaoM");
85                  YA = rsx.getDouble("Velocidade");
86                  YA2 = rsx.getDouble("Posicao");
87
88                  myHour = rsx.getTime("horaM");
89                  myHour = formatter.parse(myHour.toString());
90
91                  if ((Math.abs(myHour.getTime()-Time.getTime())/(1000))>1){
92
93                      String result = rsx.getDouble("Velocidade") + "\t\t"
94                      + rsx.getDouble("Posicao") + "\t\t" +
95                       rsx.getDouble("VelocidadeM")+
96                       "\t\t" + rsx.getDouble("PosicaoM") +
97                        "\t\t" + rsx.getDate("Data")+
98                        "\t" + rsx.getTime("Hora") + "\n";
99                      textArea1.append(result);
100
101                     Time.setTime(myHour.getTime());
102                     String join = "";
103                     if (op == 1) {
104                         join = MinAxis(Time, YM, YM2);
105                     } else if (op == 2) {
106                         join = HourAxis(Time, YM, YM2);
107                     }
108                     GoalsA.add(Double.parseDouble(join), YA);
109                     GoalsA2.add(Double.parseDouble(join),
110                      YA2 * 2 * Math.PI / 360);
111
112                     GoalsM.add(Double.parseDouble(join), YM);
113                     GoalsM2.add(Double.parseDouble(join),
114                     YM2 * 2 * Math.PI / 360);
115
116                     if (op == 1) { // Se op=1 o domínio é Minuto -
117                     //Foi definido uma determinada hora
118
119                         join = MinAxis(Time, YA, YA2);
120                     } else if (op == 2) { // Se op=2 o domínio é Hora -
121                     // Apenas foi definida uma determinada data
122
123                         join = HourAxis(Time, YA, YA2);
124                     }
125
126                     GoalsA.add(Double.parseDouble(join), YA);
127                     GoalsA2.add(Double.parseDouble(join),
128                     YA2 * 2 * Math.PI / 360);
129
130                 }
131             }
```

```
132         }
133         xyDataset.addSeries(GoalsA);
134         xyDataset.addSeries(GoalsA2);
135         xyDataset.addSeries(GoalsM);
136         xyDataset.addSeries(GoalsM2);
137 
138     }
139 //  Seleciona da Base de dados e adiciona ao grafico os dados de Velocidade e
140  Direcao do vento recebidos da estacao Meteorológica.
141 
142     void MeteoH(int op, ResultSet rsx, java.sql.Time Time, double YM,
143      double YM2, DateFormat formatter) throws SQLException, ParseException {
144         textArea1.append("␣Wind␣Vel.\t\t␣Wind␣Dir.\t\t␣Data\t\t␣Hora\t\n␣");
145 
146         while (rsx.next()) {
147             if (rsx.getDate("DataM") == null) {
148 
149                 String result = rsx.getDouble("VelocidadeM") + "\t\t" +
150                  rsx.getDouble("PosicaoM") + "\t\t␣-␣\t\t␣-␣\n";
151                 textArea1.append(result);
152 
153             } else {
154 
155                 YM = rsx.getDouble("VelocidadeM");
156                 YM2 = rsx.getDouble("PosicaoM");
157 
158                 myHour = rsx.getTime("horaM");
159                 myHour = formatter.parse(myHour.toString());
160 
161                 if ((Math.abs(myHour.getTime()-Time.getTime())/(1000))>1){
162 
163                     String result = YM + "\t\t" + YM2 + "\t\t" +
164                     rsx.getDate("DataM") + "\t" + rsx.getTime("HoraM") + "\n";
165                     textArea1.append(result);
166 
167                     Time.setTime(myHour.getTime());
168                     String join = "";
169                     if (op == 1) {
170                         join = MinAxis(Time, YM, YM2);
171                     } else if (op == 2) {
172                         join = HourAxis(Time, YM, YM2);
173                     }
174 
175                     GoalsM.add(Double.parseDouble(join), YM);
176                     GoalsM2.add(Double.parseDouble(join),YM2*2*Math.PI/360);
177                 }
178 
179             }
180         }
181         xyDataset.addSeries(GoalsM);
```

```
182
183         xyDataset.addSeries(GoalsM2);
184
185     }
186 // Seleciona da Base de dados e adiciona ao gráfico os dados de Velocidade
187 //e Direcao do aerogerador.
188
189     void AeroH(int op, ResultSet rsx, java.sql.Time Time, double YA,
190     double YA2, DateFormat formatter) throws SQLException, ParseException {
191
192         textArea1.append("AeroVel.\t\t␣AeroPos.\t\t␣␣Data\t\t␣Hora\t\n␣");
193         while (rsx.next()) {
194             if (rsx.getDate("Data") == null) {
195
196                 String result = rsx.getDouble("Velocidade") + "\t\t" +
197                  rsx.getDouble("Posicao") + "\t\t␣-␣\t\t␣-␣\n";
198                 textArea1.append(result);
199
200             } else {
201
202                 YA = rsx.getDouble("Velocidade");
203                 YA2 = rsx.getDouble("Posicao");
204
205                 myHour = rsx.getTime("hora");
206                 myHour = formatter.parse(myHour.toString());
207                 if ((Math.abs(myHour.getTime()-Time.getTime())/(1000))>1){
208
209                     String result = YA + "\t\t" + YA2 + "\t\t" +
210                     rsx.getDate("Data") + "\t" + rsx.getTime("Hora") + "\n";
211
212                     textArea1.append(result);
213                     Time.setTime(myHour.getTime());
214
215                     String join = "";
216                     if (op == 1) {
217
218                         join = MinAxis(Time, YA, YA2);
219                     } else if (op == 2) {
220                         join = HourAxis(Time, YA, YA2);
221                     }
222                     GoalsA.add(Double.parseDouble(join), YA);
223                     GoalsA2.add(Double.parseDouble(join),
224                     YA2 * 2 * Math.PI / 360);
225                 }
226             }
227         }
228         xyDataset.addSeries(GoalsA);
229         xyDataset.addSeries(GoalsA2);
230
231     }
```

```
232 // Acerta o dominio como minutos ( Op=1)
233
234     String MinAxis(java.sql.Time Time, double Y, double Y2) {
235         chart.getXYPlot().getDomainAxis().setLabel("Minuto");
236
237         String[] MinSec = Time.toString().split(":");
238         int mins = Integer.parseInt(MinSec[1]);
239         int sec = Integer.parseInt(MinSec[2]);
240         sec = sec * 9 / 59;
241
242         String join = String.valueOf(mins) + "." + String.valueOf(sec);
243         return join;
244     }
245
246 // Acerta o dominio como hora ( Op=2)
247     String HourAxis(java.sql.Time Time, double Y, double Y2) {
248         chart.getXYPlot().getDomainAxis().setLabel("Hora");
249
250         String[] hourMin = Time.toString().split(":");
251         int hour = Integer.parseInt(hourMin[0]);
252         int min = Integer.parseInt(hourMin[1]);
253         int sec = Integer.parseInt(hourMin[2]);
254
255         min = min * 9 / 59;
256         sec = sec * 9 / 59;
257
258         String join = String.valueOf(hour) + "." +
259          String.valueOf(min) + String.valueOf(sec);
260         return join;
261     }
262
263     // Seleciona da Base de dados de Velocidade e Direcao do
264     // vento e do aerogerador que tenham a data e hora em comum
265
266     void MixedNOp(ResultSet rsx) throws SQLException {
267         while (rsx.next()) {
268             String result = rsx.getDouble("Velocidade") + "\t\t" +
269              rsx.getDouble("Posicao") + "\t\t" + rsx.getDouble("VelocidadeM")
270               + "\t\t" + rsx.getDouble("PosicaoM") + "\t\t" +
271                rsx.getDate("Data") + "\t" + rsx.getTime("Hora") + "\n";
272
273             textArea1.append(result);
274         }
275     }
276
277     // Seleciona da Base de dados de Velocidade e Direcao do aerogerador
278
279     void AeroNOp(ResultSet rsx) throws SQLException {
280         while (rsx.next()) {
281             String result = rsx.getDouble("Velocidade") + "\t\t" +
```

```
282             rsx.getDouble("Posicao") + "\t\t" + rsx.getDate("Data")
283             + "\t" + rsx.getTime("Hora") + "\n";
284 
285             textArea1.append(result);
286         }
287     }
288 
289     // Seleciona da Base de dados de Velocidade e Direcao do vento
290     void MeteoNOp(ResultSet rsx) throws SQLException {
291         while (rsx.next()) {
292             String result = rsx.getDouble("VelocidadeM") + "\t\t" +
293              rsx.getDouble("PosicaoM") + "\t\t" + rsx.getDate("DataM")
294              + "\t" + rsx.getTime("HoraM") + "\n";
295 
296             textArea1.append(result);
297         }
298     }
299     // Imprime os dados seleciona da Base de dados
300     void printDB(int opp, ResultSet rsx) throws SQLException, ParseException {
301 
302         GoalsA.clear();
303         GoalsA2.clear();
304         GoalsM.clear();
305         GoalsM2.clear();
306         xyDataset.removeAllSeries();
307 
308         DateFormat formatter = new SimpleDateFormat("HH:mm:ss");
309         String DateX = "00:10:00";
310 
311         java.util.Date myHourIn = formatter.parse(DateX);
312         java.sql.Time Time = new java.sql.Time(myHourIn.getTime());
313         if ((!jCheckBox3.isSelected() && jCheckBox6.isSelected()) ||
314          (jCheckBox3.isSelected() && jCheckBox6.isSelected())) {
315 
316             if (jComboBox1.getSelectedIndex() == 3) {
317                 MixedH(opp, rsx, Time, YA, YA2, YM, YM2, formatter);
318             } else if (jComboBox1.getSelectedIndex() == 1) {
319                 MeteoH(opp, rsx, Time, YM, YM2, formatter);
320             } else if (jComboBox1.getSelectedIndex() == 0) {
321                 AeroH(opp, rsx, Time, YA, YA2, formatter);
322             }
323         } else if (!jCheckBox3.isSelected() && !jCheckBox6.isSelected()) {
324 
325             if (jComboBox1.getSelectedIndex() == 3) {
326                 MixedNOp(rsx);
327             } else if (jComboBox1.getSelectedIndex() == 1) {
328                 MeteoNOp(rsx);
329             } else if (jComboBox1.getSelectedIndex() == 0) {
330                 AeroNOp(rsx);
331             }
```

```
332
333         } else if (jCheckBox3.isSelected() && !jCheckBox6.isSelected()) {
334             if (jComboBox1.getSelectedIndex() == 3) {
335                 MixedH(opp, rsx, Time, YA, YA2, YM, YM2, formatter);
336             } else if (jComboBox1.getSelectedIndex() == 1) {
337                 MeteoH(opp, rsx, Time, YM, YM2, formatter);
338             } else if (jComboBox1.getSelectedIndex() == 0) {
339                 AeroH(opp, rsx, Time, YA, YA2, formatter);
340             }
341         }
342     }
343
344 // Botao "Pesquisar" - Valida os dados de data e hora e imprime na textArea.
345     private void button1ActionPerformed(java.awt.event.ActionEvent evt) {
346         // TODO add your handling code here:
347         System.out.println("Counter␣:" + jComboBox1.getSelectedIndex()
348         + "␣-␣" + jComboBox1.getItemAt(jComboBox1.getSelectedIndex()));
349
350         textArea1.setText("␣");
351         try {
352 // Verifica se foi especificada uma data, se esta e valida, e prepara
353 // a String "Where" para a query.
354
355             if ((!jTextField3.getText().isEmpty()) &&
356             (!jTextField4.getText().isEmpty()) &&
357             (!jTextField5.getText().isEmpty()) &&
358             (jTextField3.getText() != null) &&
359             (jTextField5.getText() != null) &&
360             (jTextField4.getText() != null)) {
361
362                 System.out.println(dateValidation(jTextField3.getText() +
363                  "-" + jTextField5.getText() + "-" +
364                  jTextField4.getText()) == true);
365
366                 if (jCheckBox3.isSelected() &&
367                 dateValidation(jTextField3.getText() + "-" +
368                  jTextField5.getText() + "-" +
369                  jTextField4.getText()) == true) {
370
371                     DateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
372                     String DateX = jTextField4.getText() + "-" +
373                     jTextField5.getText() + "-" + jTextField3.getText();
374
375                     myDate = formatter.parse(DateX);
376                     sqlDate = new java.sql.Date(myDate.getTime());
377                     if (jComboBox1.getSelectedIndex() == 3 ||
378                      jComboBox1.getSelectedIndex() == 0) {
379
380                         data = "␣WHERE␣Data␣=?␣";
381                     } else if (jComboBox1.getSelectedIndex() == 1) {
```

```
382                         data = "␣WHERE␣DataM␣=?␣";
383                     }
384                     op = true;
385                     d = true;
386                 } else {
387                     limparData();
388                 }
389             } else {
390                 limparData();
391             }
392 // Verifica se foi especificada uma hora, se esta é valida, e
393 //prepara a String "Where" ou a extensao da mesma "And" para a query.
394 
395             if (jTextField6.isEnabled() && (!jTextField6.getText().isEmpty())
396              && (jTextField6.getText() != null) &&
397              !jTextField6.getText().matches("^[a-zA-Z]+$")) {
398 
399                 if (Integer.parseInt(jTextField6.getText()) < 24 &&
400                  Integer.parseInt(jTextField6.getText()) >= 0) {
401 
402                     if (jCheckBox6.isSelected()) {
403                         if (Integer.parseInt(jTextField6.getText()) < 24 &&
404                          Integer.parseInt(jTextField6.getText()) >= 0) {
405 
406                             if (d == true &&
407                             dateValidation(jTextField3.getText()
408                              + "-" + jTextField5.getText() + "-" +
409                               jTextField4.getText()) == true) {
410 
411                                 if (jComboBox1.getSelectedIndex() == 3) {
412                                     hora = "and␣EXTRACT(HOUR␣from␣" +
413                                     "␣aerodata.Hora)␣=␣?␣" +
414                                     "␣and␣EXTRACT(HOUR␣from␣" +
415                                     "␣meteodata.HoraM)␣=␣?␣";
416 
417                                 } else if(jComboBox1.getSelectedIndex()==0){
418                                     hora = "and␣EXTRACT(HOUR␣from␣" +
419                                     "␣aerodata.Hora)␣=␣?";
420 
421                                 } else if(jComboBox1.getSelectedIndex()==1) {
422                                     hora = "and␣EXTRACT(HOUR␣from␣" +
423                                     "␣meteodata.HoraM)=?";
424                                 }
425                                 op = true;
426                                 h = true;
427 
428                             } else if ((d == false ||
429                             dateValidation(jTextField3.getText()
430                              + "-" + jTextField5.getText() + "-" +
431                              jTextField4.getText()) == false) &&
```

```
                            !jCheckBox3.isSelected()) {

                                limparData();
                                h = true;
                                if (jComboBox1.getSelectedIndex() == 3) {
                                    hora = "WHERE␣EXTRACT(HOUR␣from" +
                                    "␣aerodata.Hora)␣=␣?␣and" +
                                    "␣EXTRACT(HOUR␣from␣"+
                                    "␣meteodata.HoraM)␣=␣?␣";

                                } else if(jComboBox1.getSelectedIndex()==0){

                                    hora = "WHERE␣EXTRACT(HOUR␣from" +
                                    "␣aerodata.Hora)␣=␣?␣";
                                } else if(jComboBox1.getSelectedIndex()==1){
                                    hora = "WHERE␣EXTRACT(HOUR␣from" +
                                    "␣meteodata.HoraM)␣=␣?␣";
                                }
                                op = true;

                            }
                        } else {
                            limparHora();
                        }
                    }
                } else {
                    limparHora();
                }
            } else {
                limparHora();
            }

            conn = DriverManager.getConnection(DB_URL, USER, PASS);

// Verifica se foi especificada uma data/hora, prepara a query completa,
 executa-a e imprime o resultado na textArea

            if (d == true || h == true || jCheckBox3.isSelected() ||
             jCheckBox6.isSelected()) {

                if (op == true) {

                    if (jComboBox1.getSelectedIndex() == 3) {

                        sql = "SELECT␣*␣from␣mysql.aerodata,␣mysql.meteodata"
                        + data + "␣" + hora + "" + tables;

                    } else if (jComboBox1.getSelectedIndex() == 0) {
                        sql = "SELECT␣*␣from␣mysql.aerodata" + data +
                         "␣" + hora;
```

```
482
483                 } else if (jComboBox1.getSelectedIndex() == 1) {
484                     sql = "SELECT␣*␣from␣mysql.meteodata" + data +
485                     "␣" + hora;
486                 }
487
488                 //System.out.println(sql);
489                 PreparedStatement ps = conn.prepareStatement(sql);
490
491                 if (d == true && h == true) {
492                     ps.setDate(1, sqlDate);
493                     ps.setInt(2, Integer.parseInt(jTextField6.getText()));
494                       // gets hour in 12h format
495
496                     if (jComboBox1.getSelectedIndex() == 3) {
497                     ps.setInt(3,Integer.parseInt(jTextField6.getText()));
498                     }
499                     rs = ps.executeQuery();
500                     printDB(1, rs);
501                     rs.close();
502                 } else if (d == true && h == false) {
503                     ps.setDate(1, sqlDate);
504                     rs = ps.executeQuery();
505                     printDB(2, rs);
506                     rs.close();
507                 } else if (h == true && d == false) {
508
509                     ps.setInt(1, Integer.parseInt(jTextField6.getText()));
510                     if (jComboBox1.getSelectedIndex() == 3) {
511                     ps.setInt(2, Integer.parseInt(jTextField6.getText()));
512                     }
513                     rs = ps.executeQuery();
514                     printDB(0, rs);
515                     rs.close();
516                 }
517
518                 ps.close();
519             } else {
520                 System.out.println("Dia:" + jTextField4.getText() +
521                  "\t␣mes:" + jTextField5.getText() + "\t␣ano:" +
522                   jTextField4.getText());
523             }
524             op = false;
525             d = false;
526             h = false;
527
528         } else {// Caso não tenha sido especificado a data e a hora,
529          prepara a query completa, executa-a e imprime o
530          // resultado na textArea
531
```

```
532                 if (jComboBox1.getSelectedIndex() == 3) {
533 
534                     sql = "SELECT␣*␣from␣mysql.aerodata,mysql.meteodata␣" +
535                     "␣WHERE␣mysql.aerodata.Hora␣=␣mysql.meteodata.HoraM␣and␣" +
536                     "␣␣mysql.aerodata.Data=mysql.meteodata.DataM␣GROUP␣BY␣" +
537                     "␣mysql.aerodata.`Hora`␣";
538 
539                 } else if (jComboBox1.getSelectedIndex() == 0) {
540                     sql = "SELECT␣*␣from␣mysql.aerodata␣";
541                 } else if (jComboBox1.getSelectedIndex() == 1) {
542                     sql = "SELECT␣*␣from␣mysql.meteodata";
543                 }
544 
545                 // System.out.println(sql);
546                 stmt = conn.createStatement();
547                 rs = stmt.executeQuery(sql);
548                 printDB(0, rs);
549                 rs.close();
550                 stmt.close();
551 
552             }
553 
554             conn.close();
555         } catch (SQLException | ParseException ex) {
556             Logger.getLogger(Interface.class.getName()).log(Level.SEVERE,
557              null, ex);
558         }
559 
560     }
561     final static String DATE_FORMAT = "dd-MM-yyyy";
562 
563     // Valida a data: "true " se esta é valida, "false" se não é válida.
564     boolean dateValidation(String date) {
565 
566         try {
567             DateFormat df = new SimpleDateFormat(DATE_FORMAT);
568             df.setLenient(false);
569             df.parse(date);
570             return true;
571         } catch (ParseException e) {
572             return false;
573         }
574     }
575 //Botão "Limpar"
576     private void button2ActionPerformed(java.awt.event.ActionEvent evt) {
577         // TODO add your handling code here:
578         textArea1.setText("");
579 
580 
581     }
```

```
582     //CheckBox relativa a data -  ativa caso seja selecionada e
583     //desativa caso não esteja selecionada.
584     private void jCheckBox3ActionPerformed(java.awt.event
585     .ActionEvent evt) {
586 
587         // TODO add your handling code here:
588         if (jCheckBox3.isSelected()) {
589             jTextField3.setEnabled(true);
590             jTextField4.setEnabled(true);
591             jTextField5.setEnabled(true);
592             jCheckBox6.setEnabled(true);
593 
594         } else {
595             jTextField3.setEnabled(false);
596             jTextField4.setEnabled(false);
597             jTextField5.setEnabled(false);
598             jCheckBox6.setEnabled(false);
599             jTextField6.setEnabled(false);
600         }
601     }
602 //CheckBox relativa a hora -  ativa caso seja selecionada e
603 // desativa caso não esteja selecionada.
604 
605     private void jCheckBox6ActionPerformed(java.awt.
606     event.ActionEvent evt) {
607         // TODO add your handling code here:
608         if (jCheckBox6.isSelected()) {
609             jTextField6.setEnabled(true);
610 
611         } else {
612             jTextField6.setEnabled(false);
613         }
614     }
```

Apêndice C

Sistemas de Aquisição de Dados

C.1 Arduíno 1 + Módulo RF APC220

Do código disponibilizado por Github (2017), foram realizadas pequenas alterações nos ficheiros "MinIMU9AAHRS "e no "Output".

Ficheiro "MinIMU9AAHRS":

```
/*
MinIMU-9-Arduino-AHRS
Pololu MinIMU-9 + Arduino AHRS (Attitude and Heading Reference System)
Copyright (c) 2011-2016 Pololu Corporation.
http://www.pololu.com/
MinIMU-9-Arduino-AHRS is based on sf9domahrs by Doug Weibel and Jose Julio:
http://code.google.com/p/sf9domahrs/
sf9domahrs is based on ArduIMU v1.5 by Jordi Munoz and William Premerlani,
Jose Julio and Doug Weibel:
http://code.google.com/p/ardu-imu/
MinIMU-9-Arduino-AHRS is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
MinIMU-9-Arduino-AHRS is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License
for more details.
You should have received a copy of the GNU Lesser General Public License along
with MinIMU-9-Arduino-AHRS. If not, see <http://www.gnu.org/licenses/>.
*/

```

```
23 // Uncomment the following line to use a MinIMU-9 v5 or AltIMU-10 v5.
24 // Leave commented for older IMUs (up through v4).
25 //#define IMU_V5
26
27 // Uncomment the below line to use this axis definition:
28    // X axis pointing forward
29    // Y axis pointing to the right
30    // and Z axis pointing down.
31 // Positive pitch : nose up
32 // Positive roll : right wing down
33 // Positive yaw : clockwise
34
35 //Correct directions x,y,z - gyro, accelerometer, magnetometer
36 int SENSOR_SIGN[9] = {1,1,1,-1,-1,-1,1,1,1};
37
38 // Uncomment the below line to use this axis definition:
39    // X axis pointing forward
40    // Y axis pointing to the left
41    // and Z axis pointing up.
42 // Positive pitch : nose down
43 // Positive roll : right wing down
44 // Positive yaw : counterclockwise
45 //Correct directions x,y,z - gyro, accelerometer, magnetometer
46 //int SENSOR_SIGN[9] = {1,-1,-1,-1,1,1,1,-1,-1};
47
48 //tested with Arduino Uno with ATmega328 and Arduino Duemilanove with
49 // ATMega168
50
51 #include <Wire.h>
52 #include <SPI.h>
53 #include <SD.h>
54 #include <TimeLib.h>
55 File dataFile ;
56
57 // accelerometer: 8 g sensitivity
58 // 3.9 mg/digit; 1 g = 256
59
60 //this equivalent to 1G in the raw data coming from the accelerometer
61 #define GRAVITY 256
62
63 #define ToRad(x) ((x)*0.01745329252)  // *pi/180
64 #define ToDeg(x) ((x)*57.2957795131)  // *180/pi
65
66 // gyro: 2000 dps full scale
67 // 70 mdps/digit; 1 dps = 0.07
68 #define Gyro_Gain_X 0.07 //X axis Gyro gain
69 #define Gyro_Gain_Y 0.07 //Y axis Gyro gain
70 #define Gyro_Gain_Z 0.07 //Z axis Gyro gain
71
72 //Return the scaled ADC raw data of the gyro in radians for second
```

```
73 #define Gyro_Scaled_X(x) ((x)*ToRad(Gyro_Gain_X))
74
75 //Return the scaled ADC raw data of the gyro in radians for second
76 #define Gyro_Scaled_Y(x) ((x)*ToRad(Gyro_Gain_Y))
77
78 //Return the scaled ADC raw data of the gyro in radians for second
79 #define Gyro_Scaled_Z(x) ((x)*ToRad(Gyro_Gain_Z))
80
81 // LSM303/LIS3MDL magnetometer calibration constants;
82 //use the Calibrate example from
83
84 // the Pololu LSM303 or LIS3MDL library to find the
85 //right values for your board
86
87 #define M_X_MIN -1000
88 #define M_Y_MIN -1000
89 #define M_Z_MIN -1000
90 #define M_X_MAX +1000
91 #define M_Y_MAX +1000
92 #define M_Z_MAX +1000
93
94 #define Kp_ROLLPITCH 0.02
95 #define Ki_ROLLPITCH 0.00002
96 #define Kp_YAW 1.2
97 #define Ki_YAW 0.00002
98
99 /*For debugging purposes*/
100 //OUTPUTMODE=1 will print the corrected data,
101 //OUTPUTMODE=0 will print uncorrected data of the gyros (with drift)
102 #define OUTPUTMODE 1
103
104 #define PRINT_DCM 0     //Will print the whole direction cosine matrix
105 #define PRINT_ANALOGS 0 //Will print the analog raw data
106 #define PRINT_EULER 1   //Will print the Euler angles Roll, Pitch and Yaw
107
108 #define STATUS_LED 13
109
110 // Integration time (DCM algorithm)  We will run the integration loop at
111 // 50Hz if possible
112 float G_Dt=0.02;
113
114 long timer=0;   //general purpuse timer
115 long timer_old;
116 long timer24=0; //Second timer used to print values
117 int AN[6]; //array that stores the gyro and accelerometer data
118 int AN_OFFSET[6]={0,0,0,0,0,0}; //Array that stores the Offset of the sensors
119
120 int gyro_x;
121 int gyro_y;
122 int gyro_z;
```

```
123 int accel_x;
124 int accel_y;
125 int accel_z;
126 int magnetom_x;
127 int magnetom_y;
128 int magnetom_z;
129 float c_magnetom_x;
130 float c_magnetom_y;
131 float c_magnetom_z;
132 float MAG_Heading;
133
134 float Accel_Vector[3]= {0,0,0}; //Store the acceleration in a vector
135 float Gyro_Vector[3]= {0,0,0};//Store the gyros turn rate in a vector
136 float Omega_Vector[3]= {0,0,0}; //Corrected Gyro_Vector data
137 float Omega_P[3]= {0,0,0};//Omega Proportional correction
138 float Omega_I[3]= {0,0,0};//Omega Integrator
139 float Omega[3]= {0,0,0};
140
141 // Euler angles
142 float roll;
143 float pitch;
144 float yaw;
145
146 float errorRollPitch[3]= {0,0,0};
147 float errorYaw[3]= {0,0,0};
148
149 unsigned int counter=0;
150 byte gyro_sat=0;
151
152 float DCM_Matrix[3][3]= {
153   {
154     1,0,0  }
155   ,{
156     0,1,0  }
157   ,{
158     0,0,1  }
159 };
160 float Update_Matrix[3][3]={{0,1,2},{3,4,5},{6,7,8}}; //Gyros here
161
162
163 float Temporary_Matrix[3][3]={
164   {
165     0,0,0  }
166   ,{
167     0,0,0  }
168   ,{
169     0,0,0  }
170 };
171
172 void setup()
```

```
{
   SD.begin(4);


  I2C_Init();

  delay(1500);

  Accel_Init();
  Compass_Init();
  Gyro_Init();

  delay(20);

  for(int i=0;i<32;i++)    // We take some readings...
    {
    Read_Gyro();
    Read_Accel();
    for(int y=0; y<6; y++)   // Cumulate values
      AN_OFFSET[y] += AN[y];
    delay(20);
    }

  for(int y=0; y<6; y++)
    AN_OFFSET[y] = AN_OFFSET[y]/32;

  AN_OFFSET[5]-=GRAVITY*SENSOR_SIGN[5];

  //Serial.println("Offset:");
  for(int y=0; y<6; y++)
    Serial.println(AN_OFFSET[y]);

  delay(2000);

  timer=millis();
  delay(20);
  counter=0;

   Serial.begin(9600);

}
int countTime=0;
double accel=0;

void loop() //Main Loop
{
  if((millis()-timer)>=20)  // Main loop runs at 50Hz
  {
    counter++;
    timer_old = timer;
```

```
223     timer=millis();
224     if (timer>timer_old)
225     {
226 // Real time of loop run. We use this on the DCM algorithm
227 //(gyro integration time)
228       G_Dt = (timer-timer_old)/1000.0;
229       if (G_Dt > 0.2)
230         G_Dt = 0; // ignore integration times over 200 ms
231     }
232     else
233       G_Dt = 0;
234
235
236
237     // *** DCM algorithm
238     // Data adquisition
239     Read_Gyro();   // This read gyro data
240     Read_Accel();     // Read I2C accelerometer
241
242     if (counter > 5)  // Read compass data at 10Hz... (5 loop runs)
243     {
244       counter=0;
245       Read_Compass();    // Read I2C magnetometer
246       Compass_Heading(); // Calculate magnetic heading
247     }
248
249     // Calculations...
250     Matrix_update();
251     Normalize();
252     Drift_correction();
253     Euler_angles();
254     // ***
255
256     printdata();
257   }
258
259 }
```

Ficheiro "Output":

```
1  /*
2  MinIMU-9-Arduino-AHRS
3  Pololu MinIMU-9 + Arduino AHRS (Attitude and Heading Reference System)
4  Copyright (c) 2011-2016 Pololu Corporation.
5  http://www.pololu.com/
6  MinIMU-9-Arduino-AHRS is based on sf9domahrs by Doug Weibel and Jose Julio:
7  http://code.google.com/p/sf9domahrs/
8  sf9domahrs is based on ArduIMU v1.5 by Jordi Munoz and William Premerlani,
9  Jose Julio and Doug Weibel:
10 http://code.google.com/p/ardu-imu/
```

```
11 MinIMU-9-Arduino-AHRS is free software: you can redistribute it and/or modify
12 it under the terms of the GNU Lesser General Public License as published by
13 the Free Software Foundation, either version 3 of the License, or
14 (at your option) any later version.
15 MinIMU-9-Arduino-AHRS is distributed in the hope that it will be useful, but
16 WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
17 FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License
18 for more details.
19 You should have received a copy of the GNU Lesser General Public License along
20 with MinIMU-9-Arduino-AHRS. If not, see <http://www.gnu.org/licenses/>.
21 */
22 void printdata(void)
23 {
24   float rYaw=0;
25     int h= hour();
26   int m= minute();
27   int s= second();
28   int d= day();
29   int M = month();
30   int y= year();
31   time_t pctime;
32  String dataString="", D="" ;
33
34
35       #if PRINT_EULER == 1
36
37       if(ToDeg(yaw)<0){
38         rYaw=360+ToDeg(yaw);
39         }else{
40         rYaw=  ToDeg(yaw);
41         }
42         countTime++;
43
44         if(countTime>=12){ // 1 = 20 ms
45    //   c++;
46       dataString = "S␣"+ String(ToDeg(pitch))+"\t"+String(rYaw) + "\t"
47       +String(h) + "\t" +  String(m)+"\t" +   String(s)+ "\t" + String(d)+
48        "\t" +String(M)+  "\t" +  String(y) ;
49
50       Serial.println(dataString);
51       countTime=0;
52   if (dataFile) {
53           dataFile.println(dataString);
54         }
55       }
56       #endif
57 while(Serial.available()>0){ // Leitura da porta serial e sincronizacao
58   if(Serial.read()=='D'){
59
60      D=Serial.readString();
```

```
61      pctime=D.toInt();
62      setTime(pctime);
63      dataFile.close();
64     dataFile = SD.open("data.txt", FILE_WRITE) ;
65
66   }
67 }
68
69 }
```

C.2 Arduíno 2 + Módulo RF APC220

```
1  #include <SPI.h>
2  #include <SD.h>
3  #include <TimeLib.h>
4  //#include "I2Cdev.h"
5  #include "TimerOne.h" // Timer Interrupt set to 2 second for read sensors
6  #include <math.h>
7
8  #define WindSensorPin (2) // The pin location of the anemometer sensor
9  #define WindVanePin (A4) // The pin the wind vane sensor is connected to
10 #define VaneOffset 0; // define the anemometer offset from magnetic north
11
12 int VaneValue; // raw analog value from wind vane
13 int Direction; // translated 0 - 360 direction
14 int CalDirection; // converted value with offset applied
15 int LastValue; // last direction value
16
17 volatile bool IsSampleRequired;
18 // this is set true every 2.5s. Get wind speed
19
20 volatile unsigned int TimerCount;
21  // used to determine 2.5sec timer count
22
23 volatile unsigned long Rotations;
24  // cup rotation counter used in interrupt routine
25
26 volatile unsigned long ContactBounceTime;
27 // Timer to avoid contact bounce in isr
28
29 float WindSpeed; // speed miles per hour
30
31 long time=0;
32 time_t pctime;
33 int indexX;
34 int index;
35 int index2;
36 String D;
```

```
37
38 void setup() {
39
40 LastValue = 0;
41 IsSampleRequired = false;
42 TimerCount = 0;
43 Rotations = 0; // Set Rotations to 0 ready for calculations
44
45 Serial.begin(9600);
46
47 pinMode(WindSensorPin, INPUT);
48 attachInterrupt(digitalPinToInterrupt(WindSensorPin),
49 isr_rotation, FALLING);
50
51 Serial.println("Davis␣Anemometer␣Test");
52 Serial.println("Speed␣(MPH)\tKnots\tDirection\tStrength");
53
54 // Setup the timer interupt
55 Timer1.initialize(500000);// Timer interrupt every 2.5 seconds
56 Timer1.attachInterrupt(isr_timer);
57 }
58
59 void loop() {
60   int h = hour();
61   int m = minute();
62   int s = second();
63   int d = day();
64   int M = month();
65   int y = year();
66
67 getWindDirection();
68
69 // Only update the display if change greater than 5 degrees.
70 if(abs(CalDirection - LastValue) > 5) {
71 LastValue = CalDirection;
72 }
73
74 if(IsSampleRequired) {
75 // convert to mp/h using the formula V=P(2.25/T)
76 // V = P(2.25/2.5) = P * 0.9
77  WindSpeed = Rotations * 0.9*0.44704;
78  Rotations = 0; // Reset count for next sample
79  IsSampleRequired = false;
80
81  String dataString ="M␣"+String(WindSpeed)+"\t" +
82  String(CalDirection)+"\t"+String(h) + "\t" +
83  String(m)+"\t" +   String(s)+ "\t" + String(d)+
84  "\t" +String(M)+  "\t" +  String(y);
85
86  Serial.println(dataString);
```

```
87 }
88
89 while(Serial.available()>0){
90  if(Serial.read()=='D'){
91    D=Serial.readString();
92    pctime=D.toInt();
93    setTime(pctime);
94   }
95 }
96 }
97
98 void isr_timer() {
99
100 TimerCount++;
101
102 if(TimerCount == 6)
103 {
104 IsSampleRequired = true;
105 TimerCount = 0;
106 }
107 }
108
109 // This is the function that the interrupt calls to
110 //increment the rotation count
111
112 void isr_rotation() {
113
114 if((millis() - ContactBounceTime) > 15 ) {
115 // debounce the switch contact.
116
117 Rotations++;
118 ContactBounceTime = millis();
119 }
120 }
121
122 // Convert MPH to Knots
123 float getKnots(float speed) {
124 return speed * 0.868976;
125 }
126
127 // Get Wind Direction
128 void getWindDirection() {
129
130 VaneValue = analogRead(WindVanePin);
131 Direction = map(VaneValue, 0, 1023, 0, 360);
132 CalDirection = Direction + VaneOffset;
133
134 if(CalDirection > 360)
135 CalDirection = CalDirection - 360;
136
```

```
137 if(CalDirection < 0)
138 CalDirection = CalDirection + 360;
139
140 }
```

C.3 PC2 + Módulo RF APC220

```
1
2  package SerialWritePower;
3
4  import java.io.*;
5  import java.text.ParseException;
6  import java.util.Calendar;
7  import javax.comm.*;
8  import java.text.SimpleDateFormat;
9  import java.text.DateFormat;
10 import java.util.concurrent.TimeUnit;
11
12 public class PowerCentral {
13
14     public PowerCentral() {
15         super();
16     }
17
18     void connect(String portName) throws Exception {
19         CommPortIdentifier portIdentifier = CommPortIdentifier.
20         getPortIdentifier(portName);
21
22         if (portIdentifier.isCurrentlyOwned()) {
23             System.out.println("Error: Port is currently in use");
24         } else {
25             CommPort commPort = portIdentifier.open(this.getClass()
26             .getName(), 2000);
27
28             if (commPort instanceof SerialPort) {
29                 SerialPort serialPort = (SerialPort) commPort;
30                 serialPort.setSerialPortParams(9600,
31                  SerialPort.DATABITS_8,
32
33                  SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
34
35                 serialPort.setDTR(true);
36                 serialPort.setRTS(false);
37                 InputStream in = serialPort.getInputStream();
38                 OutputStream out = serialPort.getOutputStream();
39
40                 (new Thread(new SerialWriter(out))).start();
41
```

```
42              } else {
43                  System.out.println("Error: Only serial ports " +
44                  " are handled by this example.");
45              }
46          }
47      }
48
49      // Thread responsável pela transmissão de dados de sincronismo
50      public static class SerialWriter implements Runnable {
51
52          java.util.Date myDate;
53          java.util.Date myTime;
54          String dia, mes, ano, hora, minuto, segundo;
55          OutputStream out;
56          long dFinal;
57          int sample;
58          double realPower = 0, activePower = 0, root_mean_square_current = 0;
59          double  root_mean_square_voltage = 0;
60          boolean sync = true, send = false;
61          Calendar calendar = Calendar.getInstance();
62
63          long dInitial = calendar.getTimeInMillis();
64
65          public SerialWriter(OutputStream out) {
66              this.out = out;
67          }
68
69          public void run() {
70              try {
71                  Calendar calendar = Calendar.getInstance();
72                  // gets a calendar using the default time zone and locale.
73                  double[] vectorVoltage = new double[254];
74                  double[] vectorCurrent = new double[254];
75                  double PF = 0;
76
77                  String messageString = "";
78                  String line, nextFile, actualFile;
79                  nextFile = "C:\\FLUKE43B_DATA\\Data1\\rec111432.txt";
80                  actualFile = "C:\\FLUKE43B_DATA\\data1\\rec111431.txt";
81                  // String [] singleWord = new String[30];
82                  int index = 111431, startData = 0;
83                  boolean dt = false;
84                  SimpleDateFormat format1 ;
85                  format1 = new SimpleDateFormat("dd \t MM \t yyyy " +
86                  " \t HH \t mm \t ss");
87
88                  DateFormat formatter = new SimpleDateFormat("dd/MM/yyyy" +
89                  " HH:mm:ss");
90
91                  String DateX;
```

```
92                  while (sync == true) {
93                      File nfile = new File(nextFile);
94                      //System.out.println("Read new  file : "+nfile.exists());
95                      if (nfile.exists()) {
96                          File file = new File(actualFile);
97
98                          FileReader fr = new FileReader(file);
99                          BufferedReader bufferedReader;
100                         bufferedReader = new BufferedReader(fr);
101
102                         index++;
103                         while ((line = bufferedReader.readLine()) != null) {
104                             String lineWords[] = line.split("␣");
105                             if (line.contains("\"Y␣Label␣␣␣␣␣␣\"␣␣\"V\"␣"+
106                               "␣\"Y␣Label␣␣␣␣␣␣\"␣\"A\"␣␣")) {
107
108                                 startData++;
109                             } else if (startData != 0) {
110                                 startData++;
111                             }
112
113                             for (int j = 0; j < lineWords.length; j++) {
114                                 if (startData == 3) {
115                                     dt = true;
116                                     startData = 0;
117                                 }
118
119                                 String singleWord[];
120                                 singleWord[] = lineWords[j].split("\\s+");
121
122                                 for (int k = 0; k < singleWord.length; k++) {
123
124                                     if (checkDateTimePattern("MM/dd/yyyy",
125                                       singleWord[k])) {
126
127                                         System.out.println("Starting␣to␣" +
128                                         "␣read␣date!");
129
130                                         String dateArray[];
131                                         dateArray[] = singleWord[k].split("/");
132
133                                         dia = dateArray[1];
134                                         mes = dateArray[0];
135                                         ano = String.valueOf(Integer.
136                                         parseInt(dateArray[2]) + 2000);
137
138                                         System.out.println("Date␣\t␣" + dia +
139                                          "/" + mes + "/" + ano);
140
141                                     } else if (checkDateTimePattern("hh:mm:ss",
```

```
                                singleWord[k])) {

                            System.out.println("Starting␣to␣read␣" +
                            "␣time!");

                            String dateArray[];
                            dateArray[] = singleWord[k].split(":");

                            System.out.println("Time␣\t␣" +
                            singleWord[k]);
                            hora = dateArray[0];
                            minuto = dateArray[1];
                            segundo = dateArray[2];
                            DateX = dia + "/" + mes + "/" + ano +
                                "␣" + hora + ":"

                            + minuto + ":" + segundo;
                            System.out.println("Date/Time␣\t␣" +
                             DateX);

                            myDate = formatter.parse(DateX);
                            calendar.setTime(myDate);
                        }
                    }
                    if (dt) {
                        String Voltage = singleWord[2];
                        String Current = singleWord[5];

                        vectorVoltage[sample] = Double.valueOf
                        (Voltage);

                        vectorCurrent[sample] = Double.valueOf
                        (Current);

                        // System.out.println("Sample " +
                        sample);

                        if (sample == 253) {
                            PF = powerFactor(sample,
                             vectorVoltage, vectorCurrent);

                            activePower = PF;
                            sample = 0;
                            messageString = "P␣" + realPower +
                            "\t" + format1.format(calendar.
                            getTime()) + "\n";

                            System.out.println(Voltage + "\t␣" +
                            "␣-␣\t␣" +
                            root_mean_square_voltage + "\t"
```

```
192                                     + Current + "\t - \t" +
193
194                                     root_mean_square_current +
195                                     "\t" + realPower + "\t" +
196                                     PF + "\t"
197                                    + format1.format(calendar.getTime()));
198
199                                     send = true;
200                                 }
201                                 sample++;
202
203                                 // dFinal = calendar.getTimeInMillis();
204                                 if (send) {
205                                     this.out.write(messageString.
206                                     getBytes());
207
208                                     System.out.println(messageString);
209                                     System.out.println("Send data to " +
210                                     "Serial Port !");
211
212                                     send = false;
213                                     //dInitial = dFinal;
214                                     TimeUnit.SECONDS.sleep(3);
215                                 }
216
217                             }
218
219                         }
220
221                     }
222
223                     actualFile = "C:\\FLUKE43B_DATA\\data1\\rec" + index
224                      + ".txt";
225
226                     nextFile = "C:\\FLUKE43B_DATA\\data1\\rec" + (index
227                      + 1) + ".txt";
228
229                     System.out.println(actualFile);
230
231                     dt = false;
232
233                 }
234             }
235         } catch (Exception exception) {
236
237             System.out.println("IOException occured");
238             exception.printStackTrace();
239         }
240
241     }
```

```
242
243         public static boolean checkDateTimePattern(String padrao,
244         String data) {
245
246             try {
247                 SimpleDateFormat format = new SimpleDateFormat(padrao);
248                 format.parse(data);
249                 return true;
250             } catch (ParseException e) {
251                 return false;
252             }
253         }
254
255     }
256
257     public static void main(String[] args) {
258         try {
259             (new PowerCentral()).connect("COM3");
260         } catch (Exception e) {
261             // TODO Auto-generated catch block
262             e.printStackTrace();
263         }
264     }
265 }
```

APÊNDICE D

Dados adquiridos e Filtrados

Neste Capítulo estão representados os dados adquiridos e filtrados. São disponibilizadas todas as amostras adquiridas durante esta dissertação as figuras relativas a pesquisa à base de dados .

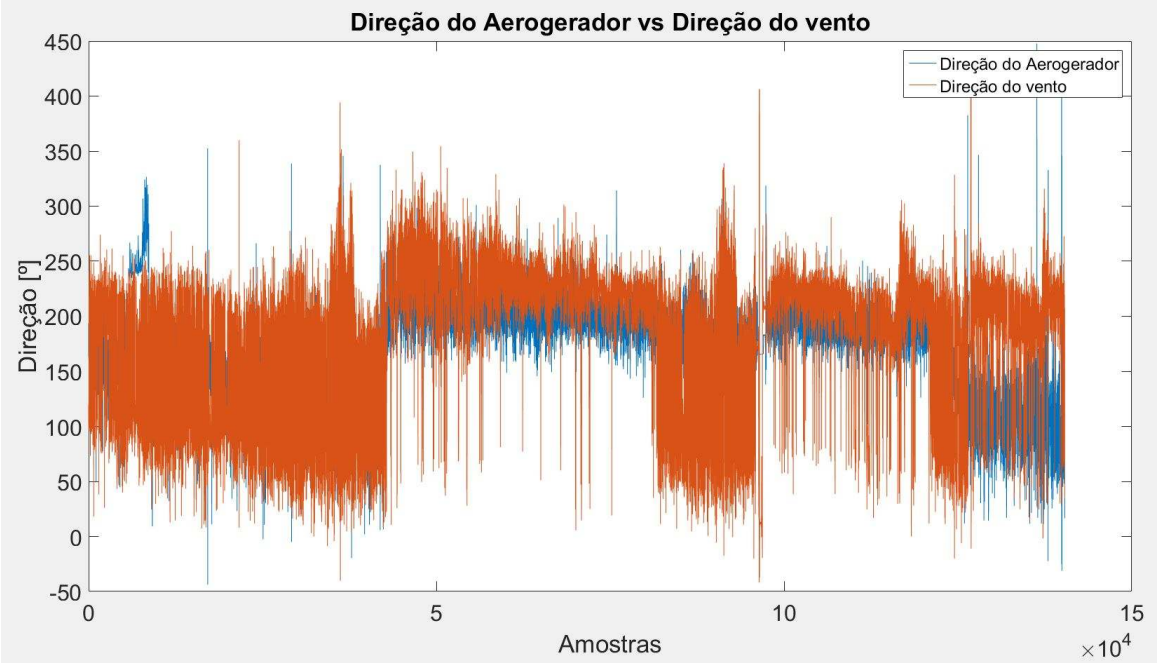


Figura D.1: Direção do vento vs Direção do aerogerador
Direção do vento Direção do aerogerador

Figura D.2: Dados do ângulo de *Yaw* do aerogerador e a aplicação do filtro de Kalman com os coeficientes [$Q = 3$ e $R = 10$]

Direção do aerogerador Direção do aerogerador filtrado

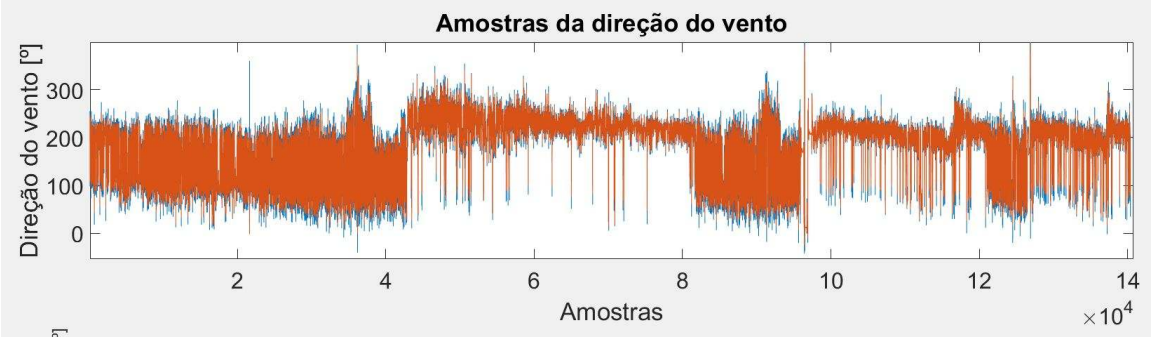


Figura D.3: Dados da direção do vento e a aplicação do filtro de Kalman com os coeficientes [$Q = 3$ e $R = 10$]

Direção do vento Direção do vento filtrado

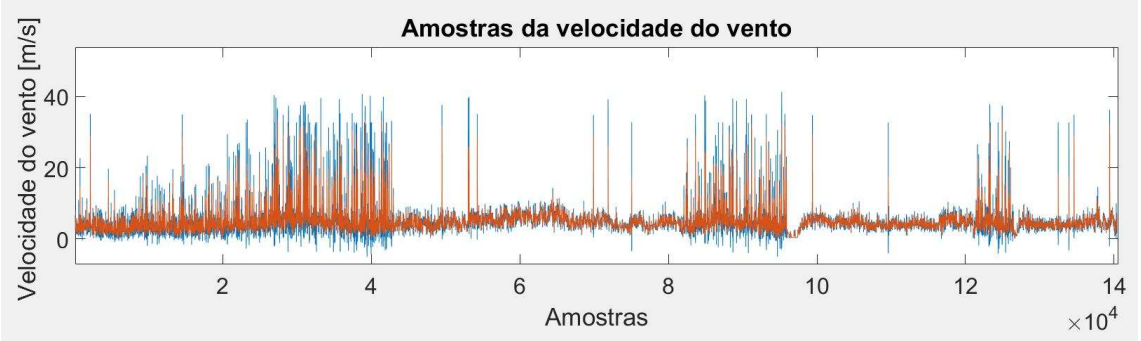


Figura D.4: Dados da velocidade do vento e a aplicação do filtro de Kalman com os coeficientes [$Q = 0.1$ e $R = 5$]

Velocidade do vento Velocidade do vento filtrado

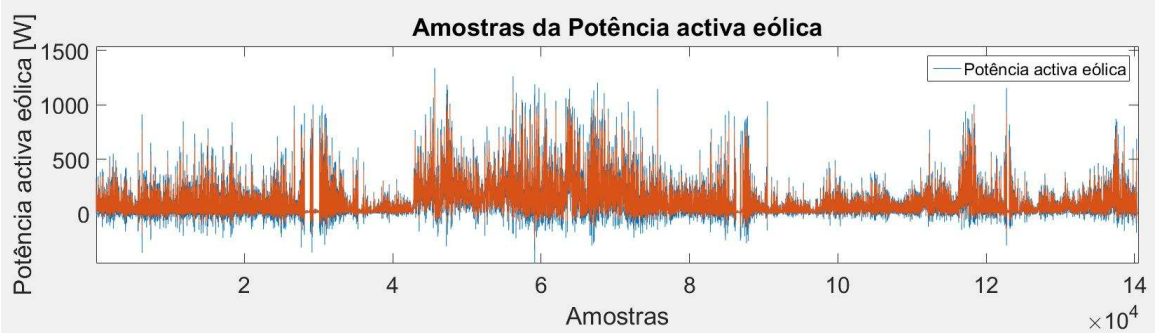


Figura D.5: Dados da produção eólica e a aplicação do filtro de Kalman com os coeficientes [$Q = 10$ e $R = 50$]

Potência elétrica produzida Potência elétrica filtrado

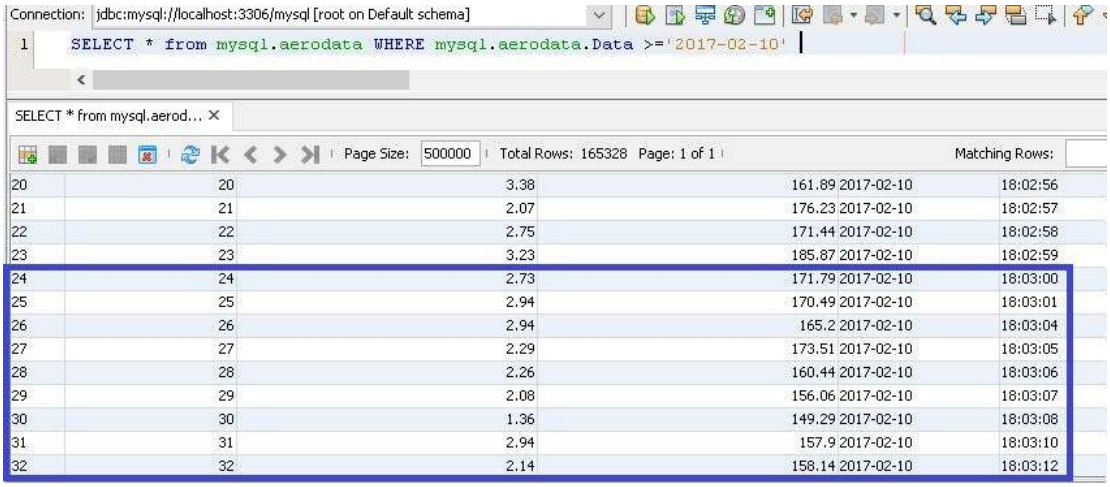


Figura D.6: Base de dados - Dados adquiridos do aerogerador, 10/02/2017

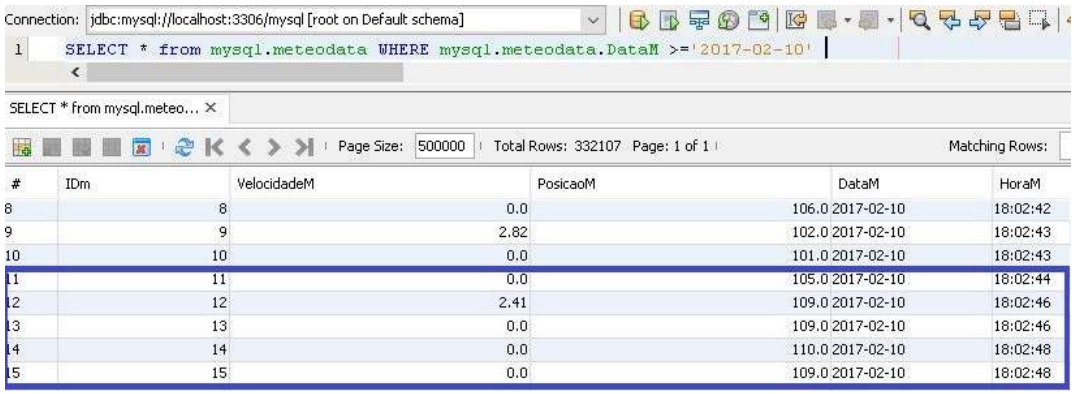
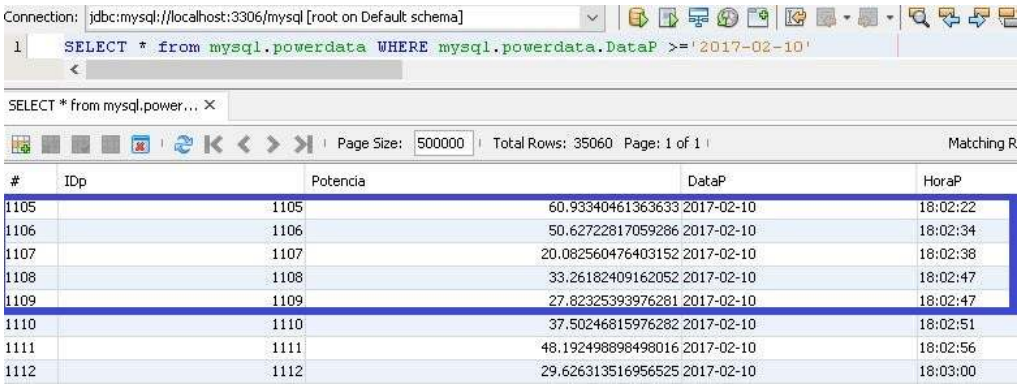


Figura D.7: Base de dados - Dados adquiridos da estação meteorológica, 10/02/2017.



#	IDp	Potencia	DataP	HoraP
1105	1105	60.93340461363633	2017-02-10	18:02:22
1106	1106	50.62722817059286	2017-02-10	18:02:34
1107	1107	20.082560476403152	2017-02-10	18:02:38
1108	1108	33.26182409162052	2017-02-10	18:02:47
1109	1109	27.82325393976281	2017-02-10	18:02:47
1110	1110	37.50246815976282	2017-02-10	18:02:51
1111	1111	48.192498898498016	2017-02-10	18:02:56
1112	1112	29.626313516956525	2017-02-10	18:03:00

Figura D.8: Base de dados - Dados da produção eólica, 10/02/2017.

Apêndice E

Modelo Ideal e Modelo Real de Potência Elétrica

Após a criação dos modelos reais e ideais de potência elétrica foram observados os resultados presentes nesta Secção, sendo aqui apresentados os gráficos com todas as amostras utilizadas.

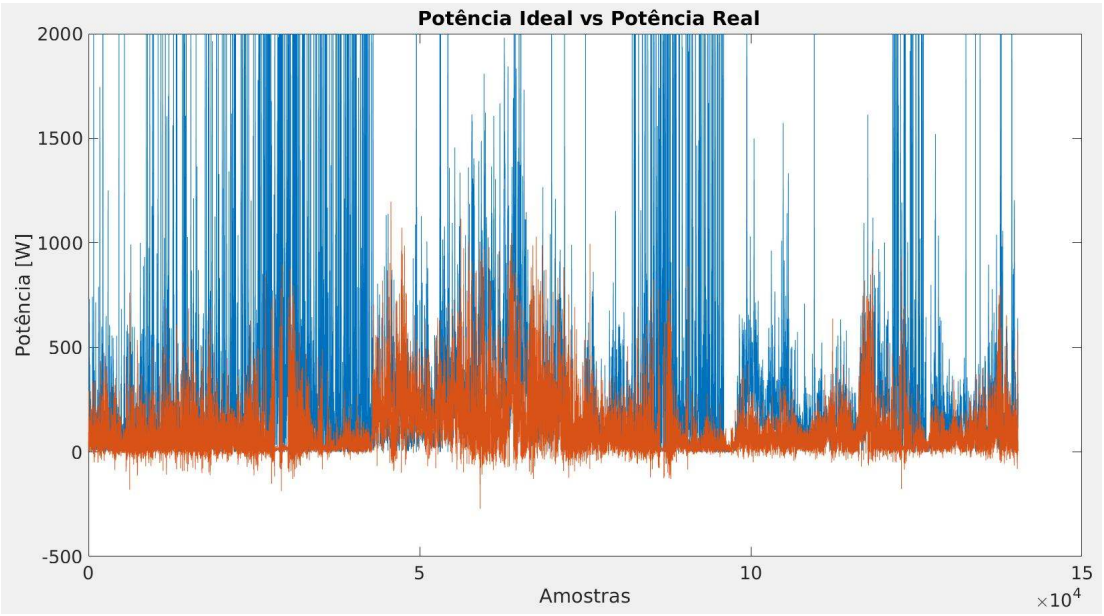


Figura E.1: Potência ideal estimada vs Potência real adquirida, Ta = 0,8 segundos.
Potência elétrica ideal (Estimada) Potência elétrica real (Adquirida)

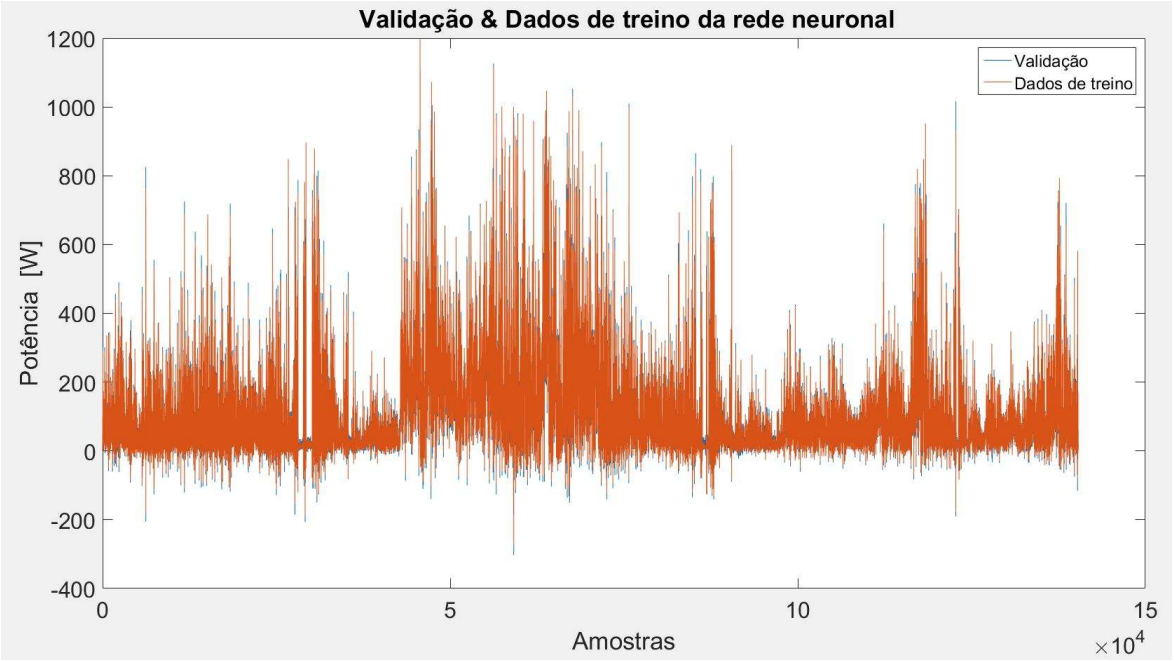


Figura E.2: Modelo Real de produção eólica criado com base nas redes neuronais, Ta = 0,8 segundos.

Validação Dados de treino

APÊNDICE F

Controladores Difusos

Neste Capítulo são apresentados os controladores difusos, tanto do binário como da posição.

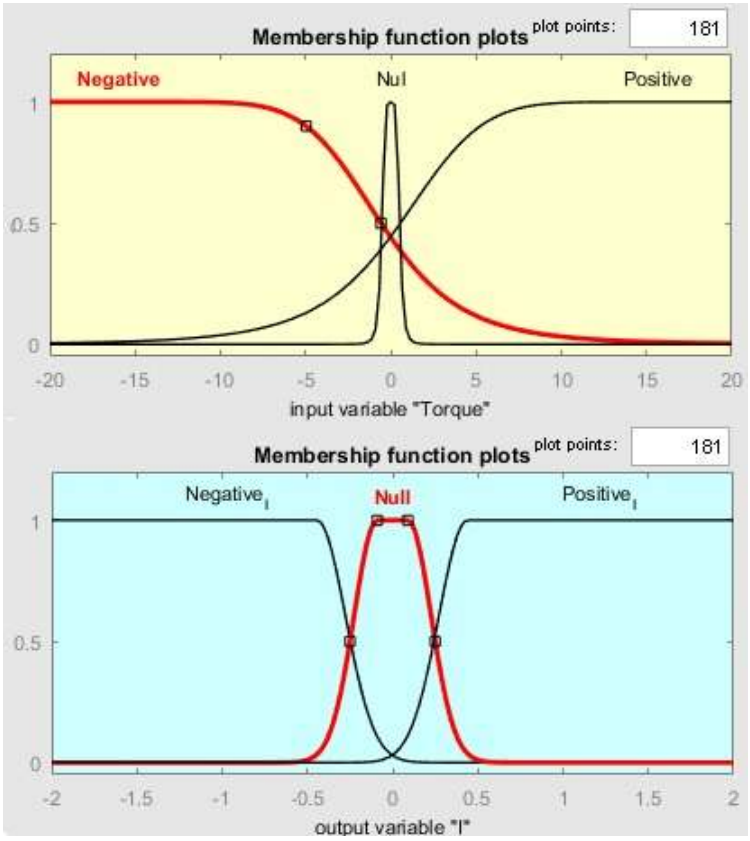


Figura F.1: Matlab/Fuzzy - Controlador difuso de Binário a utilizar a inferência de Mamdani.

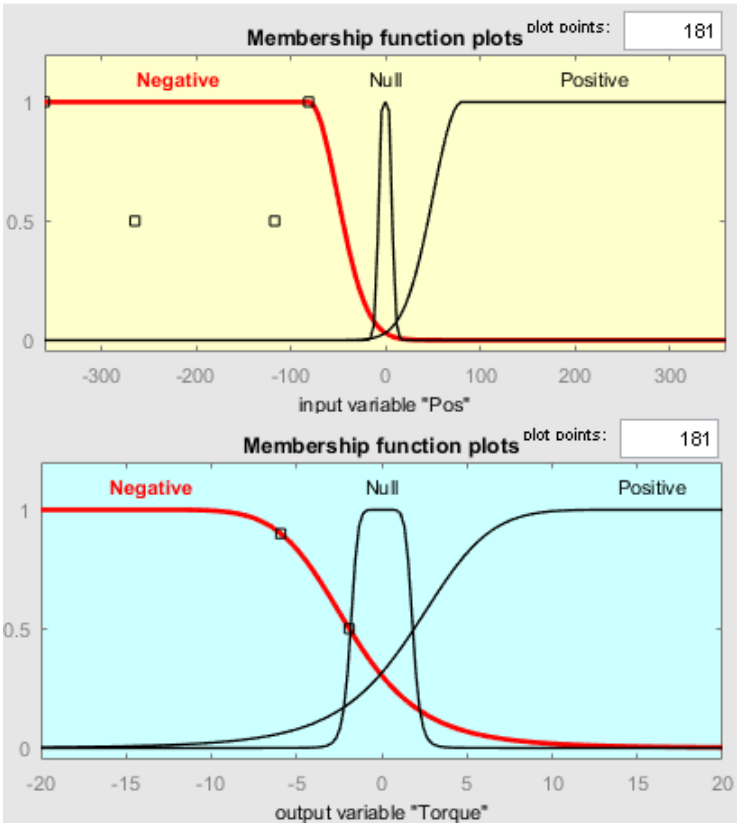


Figura F.2: Controlador difuso de Posição a utilizar a inferência de Mamdani.

APÊNDICE G

SIMULINK - DIAGRAMA DE BLOCOS DO BINÁRIO E DO ANEL DE CONTROLO COMPLETO

Neste Capítulo, estão representados os blocos mais relevantes do diagrama de blocos gerador em *MATLAB R2015b - Simulink*.

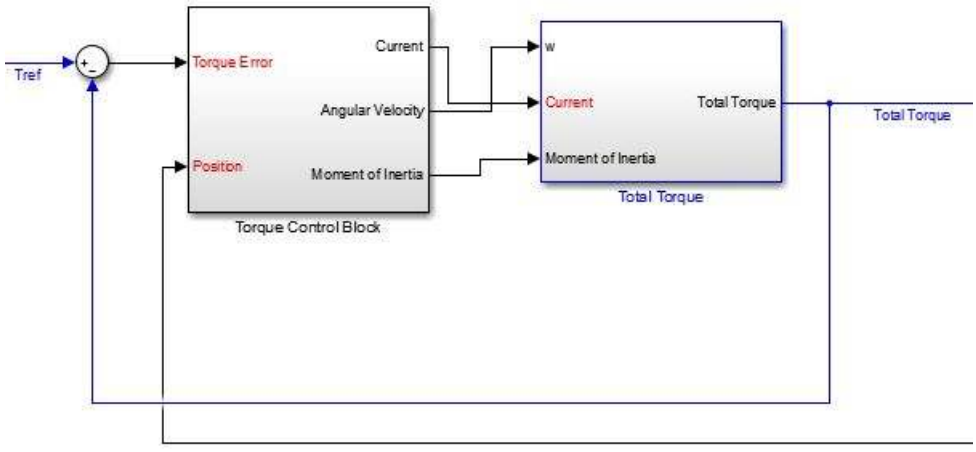


Figura G.1: Matlab/Simulink- Diagrama de blocos do controlador de Binário.

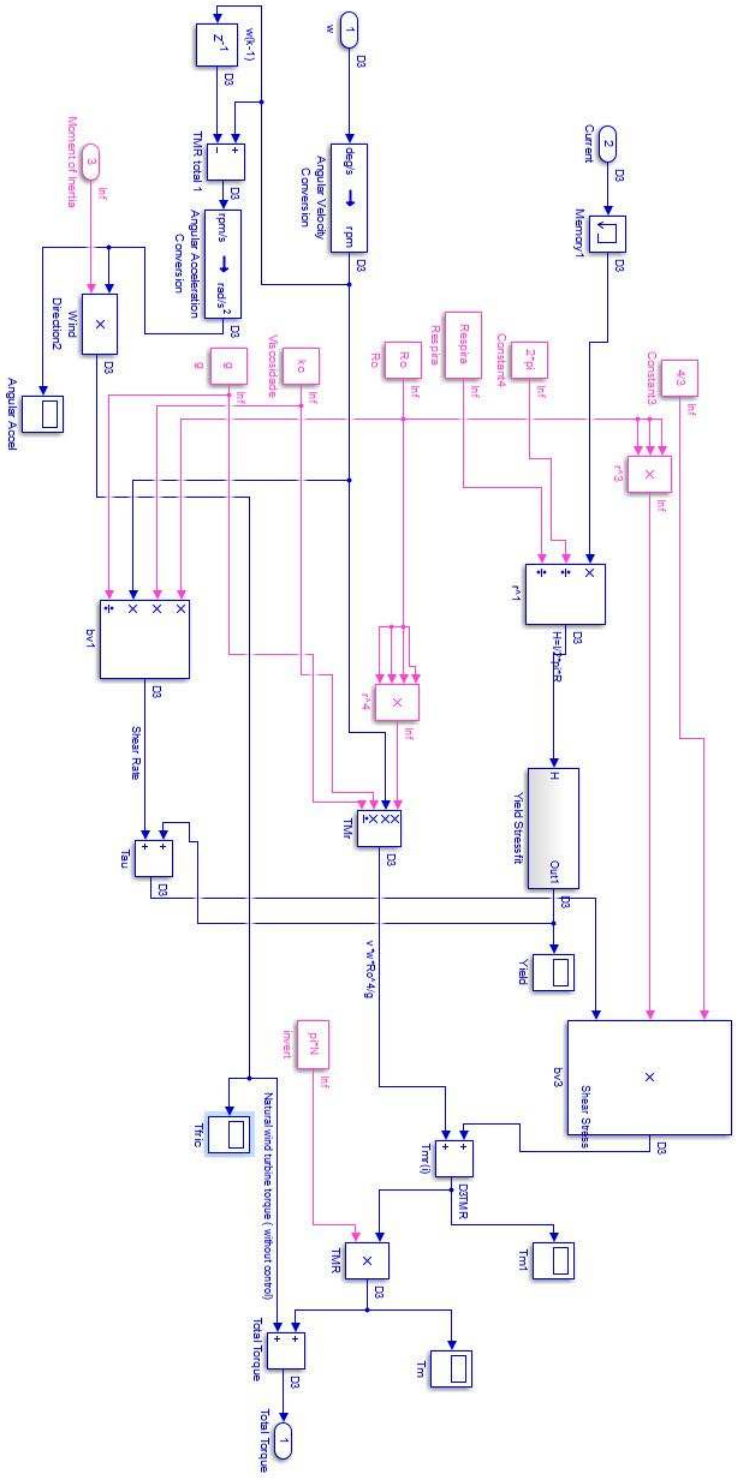


Figura G.2: Matlab/Simulink- Diagrama de blocos do binário total.

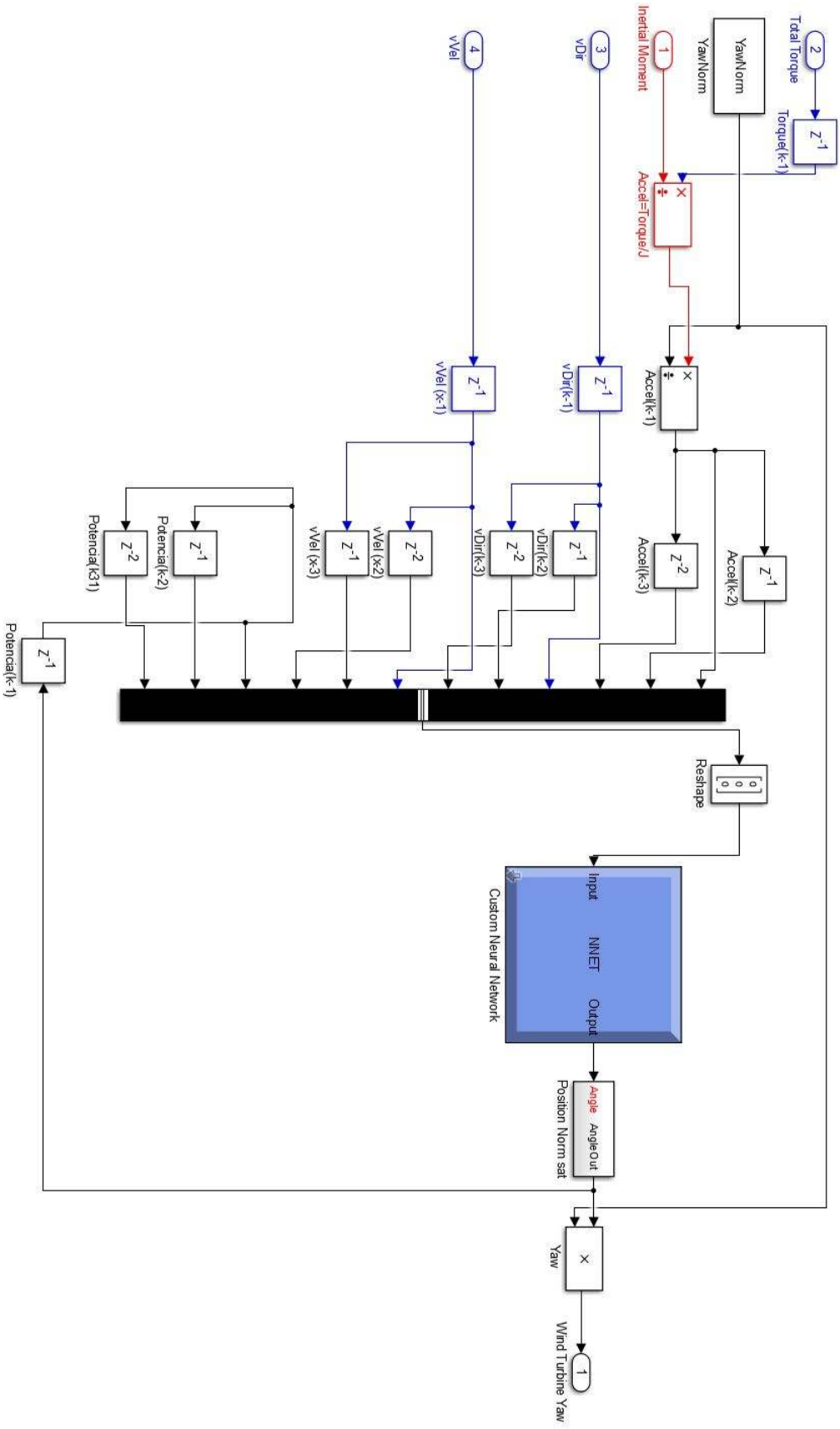


Figura G.3: Matlab/Simulink- Diagrama de blocos modelo de direção do aerogerador.

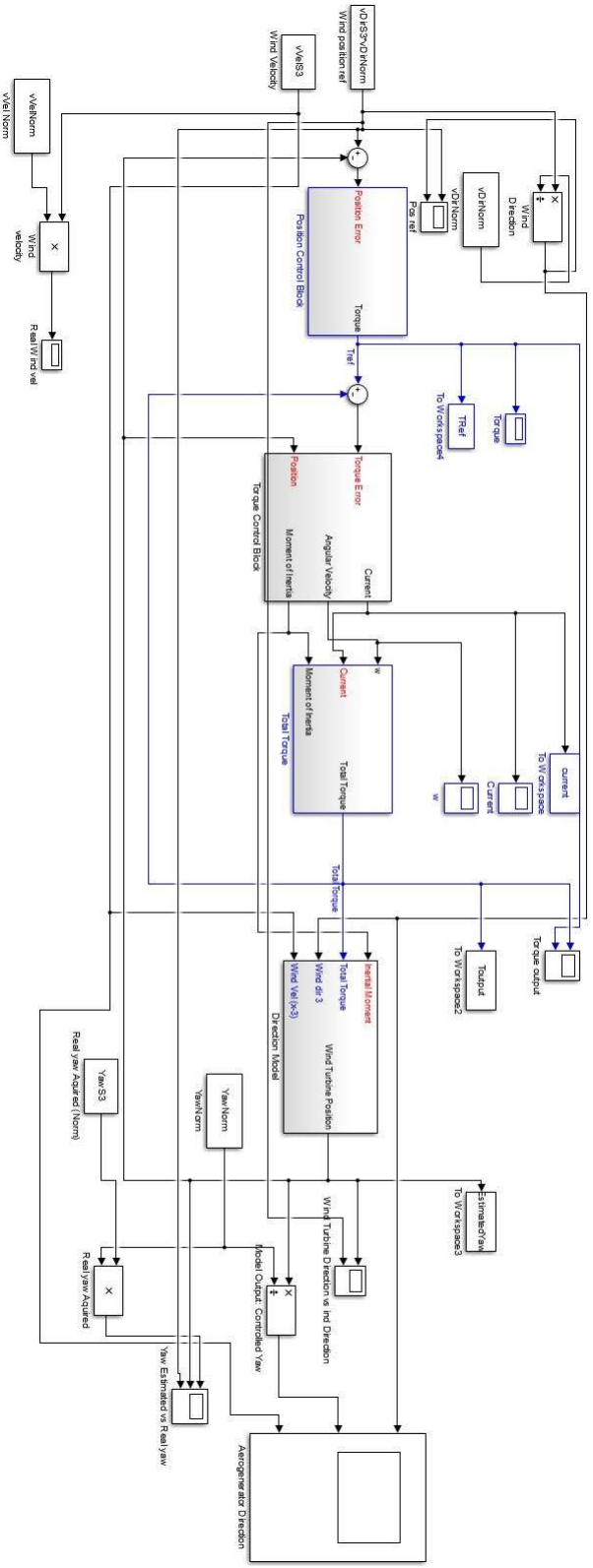


Figura G.4: Matlab/Simulink- Anel de controlo completo.

APÊNDICE H

RESULTADOS

Os resultados obtidos e discutidos no Capítulo 5 - Secção H são representados.

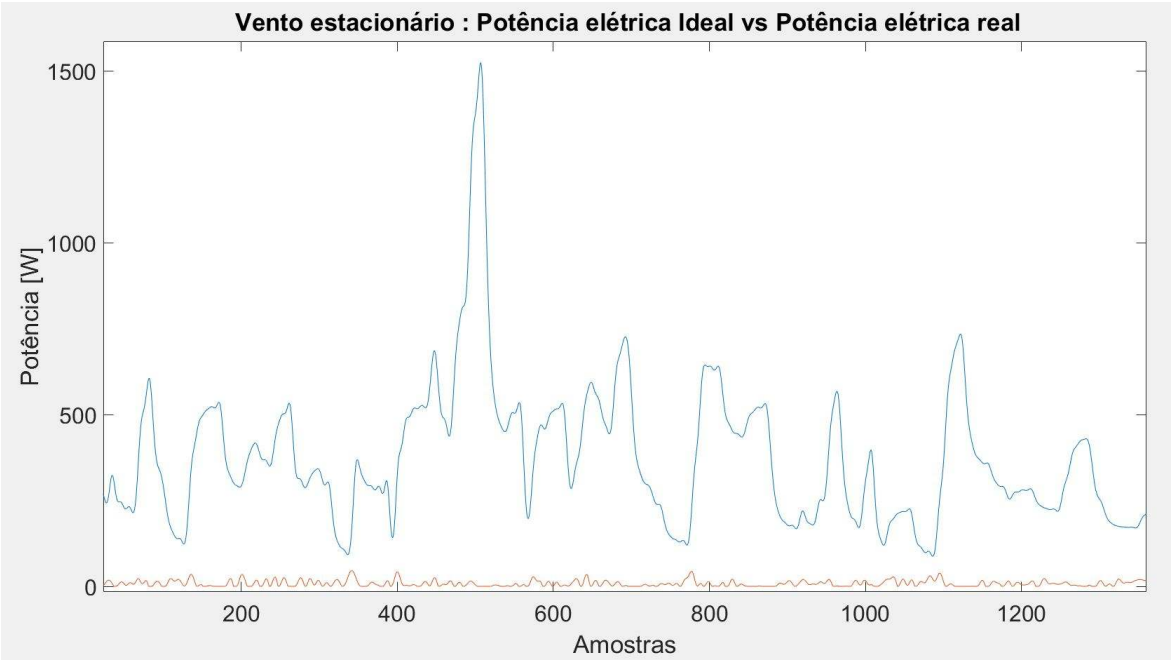


Figura H.1: Vento quase estacionário - Potência elétrica ideal vs Potência elétrica real no intervalo de [100000;101200] amostras, Ta = 0,8 segundos.

Potência elétrica ideal Potencia elétrica real

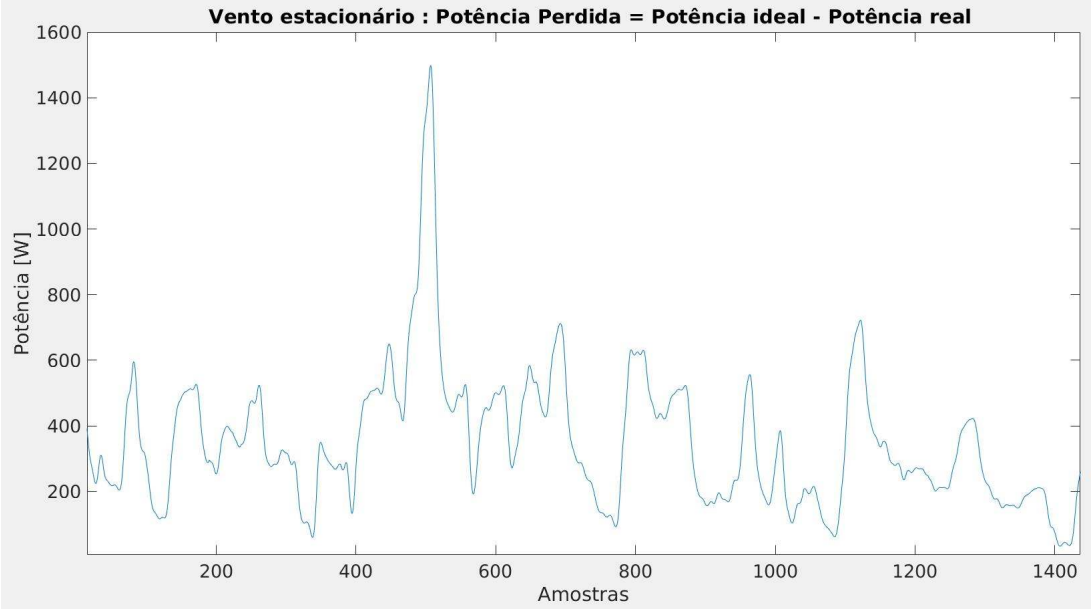


Figura H.2: Vento quase estacionário - Potência perdida no intervalo de [100000;101200] amostras, Ta = 0,8 segundos.

Potência elétrica perdida

Figura H.3: Vento quase estacionário - Direção do aerogerador controlado (*Yaw* controlado) vs Direção do aerogerador real (*Yaw* real) vs Direção do vento (1200 das 8000 amostras, Ta = 0,8 segundos.

Direção do vento Ângulo de *Yaw* controlado Ângulo de *Yaw* real

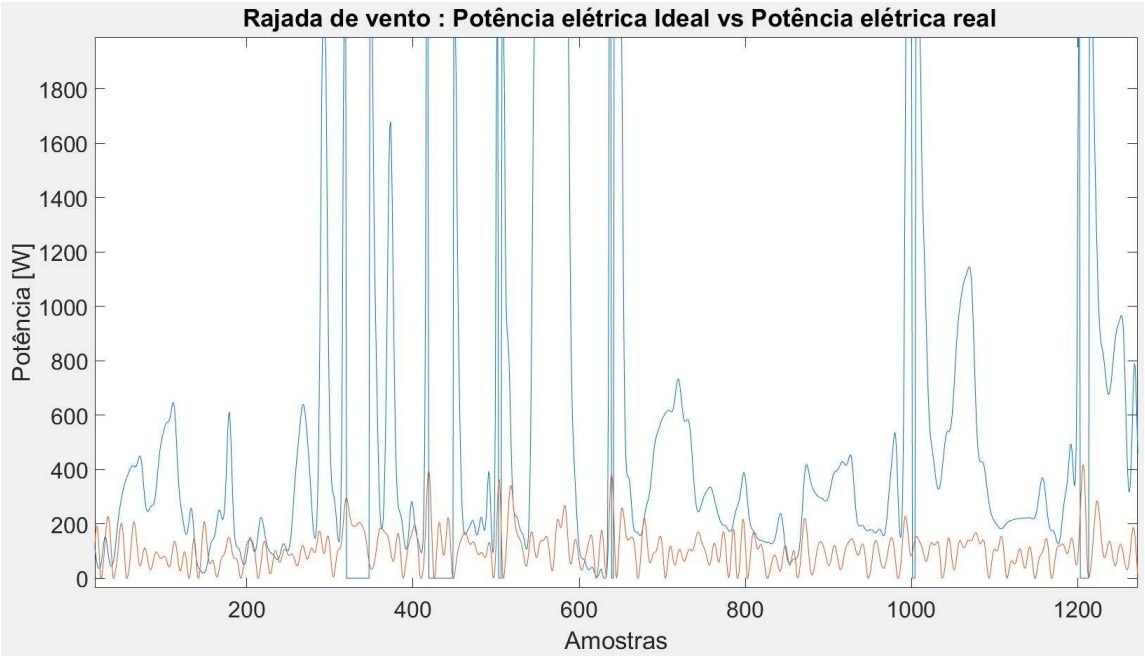


Figura H.4: Rajadas de vento - Potência elétrica ideal vs Potência elétrica real no intervalo de [32000;33200] amostras, Ta = 0,8 segundos.

Potência elétrica ideal Potência elétrica real

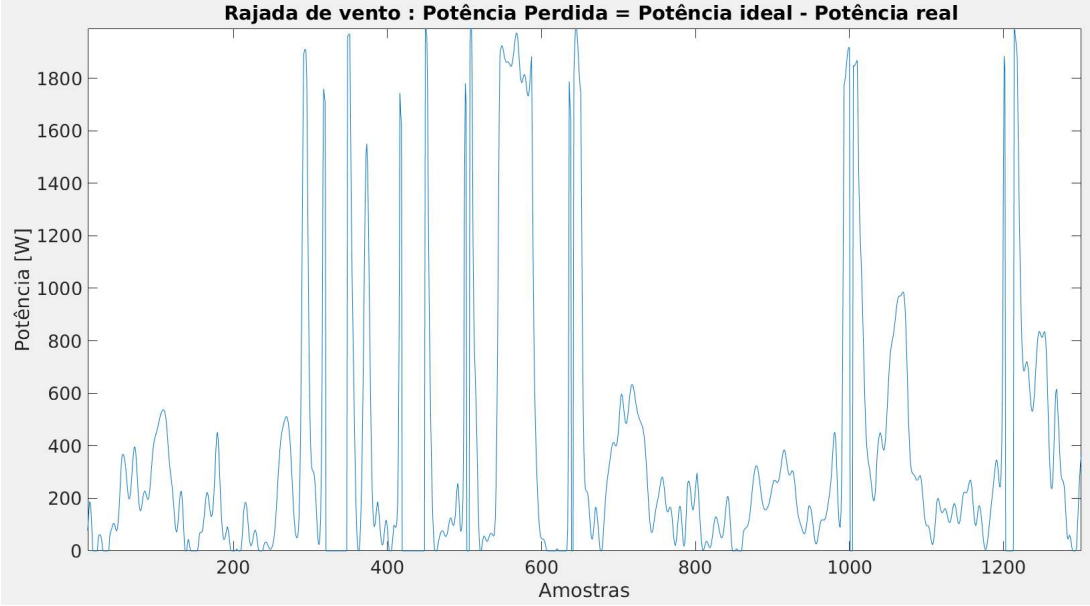


Figura H.5: Rajadas de vento - Potência perdida no intervalo de [32000;33200] amostras, Ta = 0,8 segundos.

Potência elétrica perdida

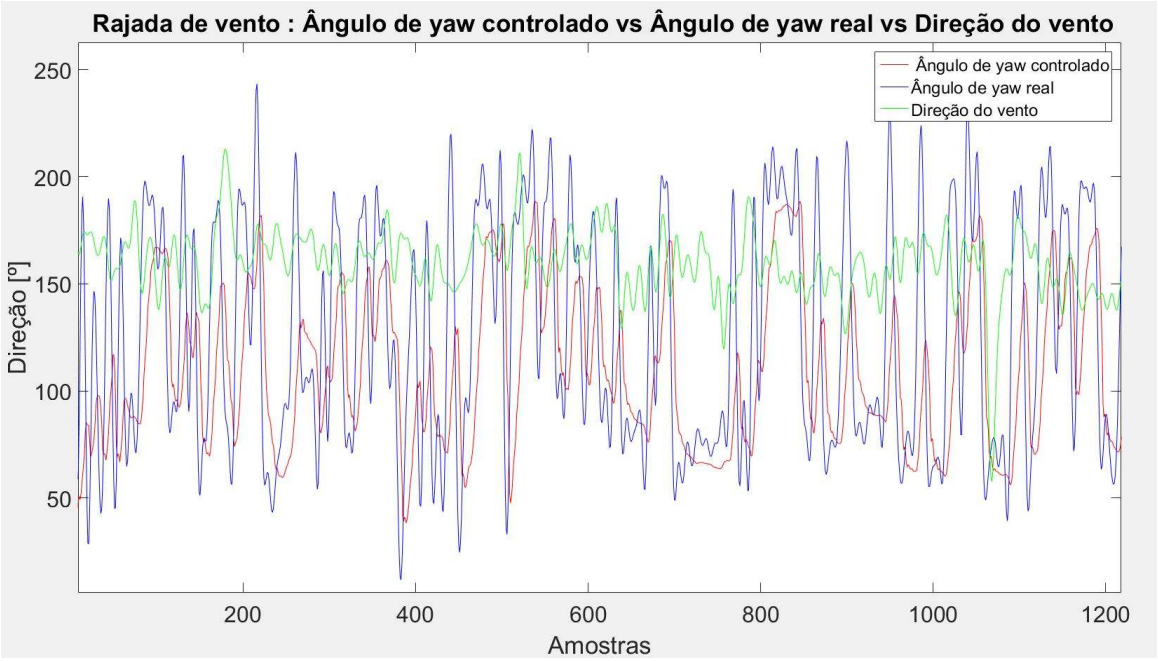


Figura H.6: Rajadas de vento - Direção do aerogerador controlado (*Yaw* controlado) vs Direção do aerogerador real (*Yaw* real) vs Direção do vento (1200 das 8000 amostras, Ta = 0,8 segundos.

Direção do vento Ângulo de *Yaw* controlado Ângulo de *Yaw* real

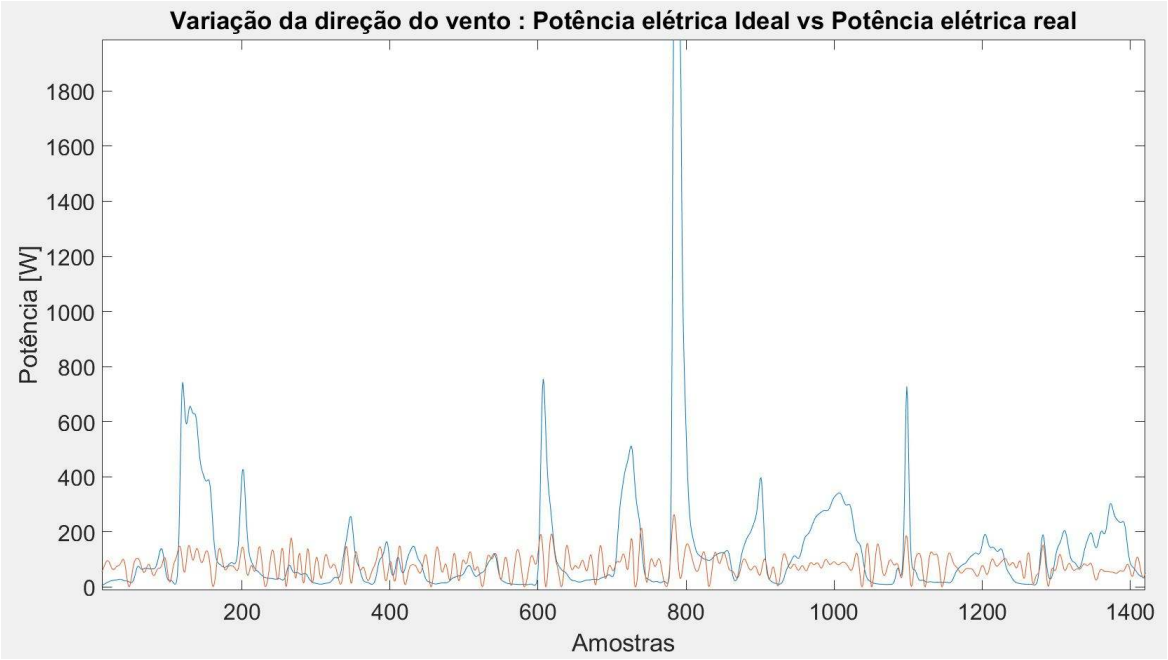


Figura H.7: Mudanças rápidas da direção do vento - Potência ideal no intervalo de amostras [1;1200], Ta = 0,8 segundos.

Potência elétrica ideal Potência elétrica real

Figura H.8: Mudanças rápidas da direção do vento - Potência perdida no intervalo de [1;1200] amostras, Ta = 0,8 segundos.

Potência elétrica perdida

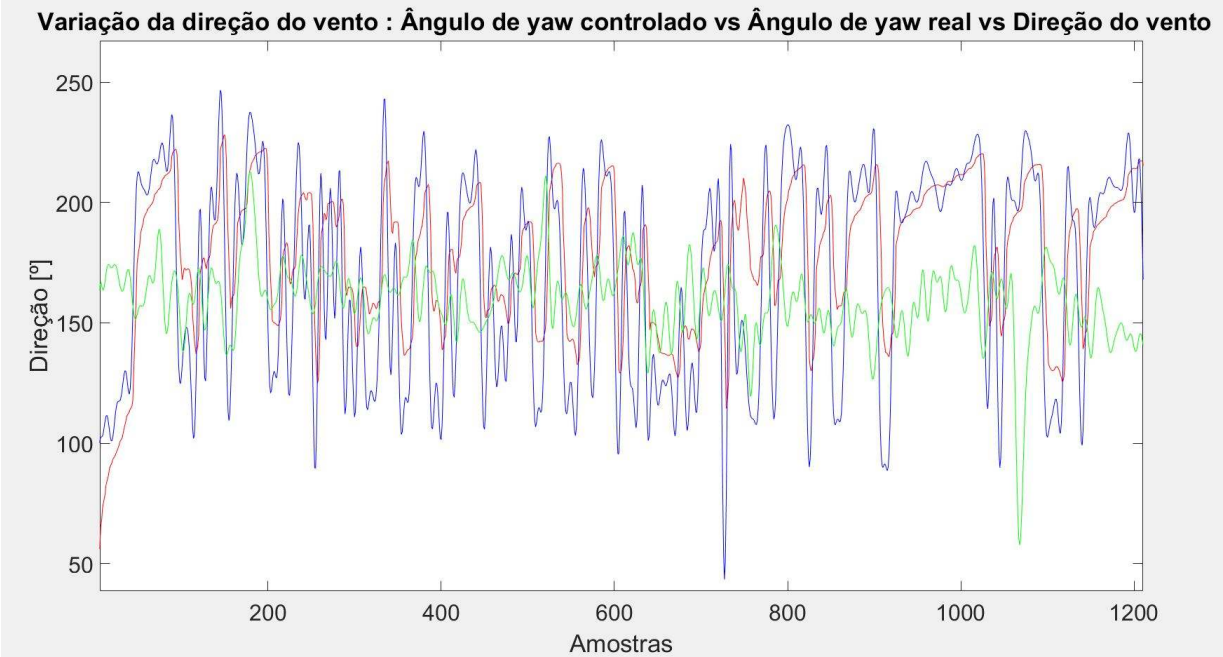


Figura H.9: Mudanças rápidas da direção do vento - Direção do aerogerador controlado (*Yaw* controlado) vs Direção do aerogerador real (*Yaw* real) vs Direção do vento (1200 das 8000 amostras, Ta = 0,8 segundos.

Direção do vento Ângulo de *Yaw* controlado Ângulo de *Yaw* real